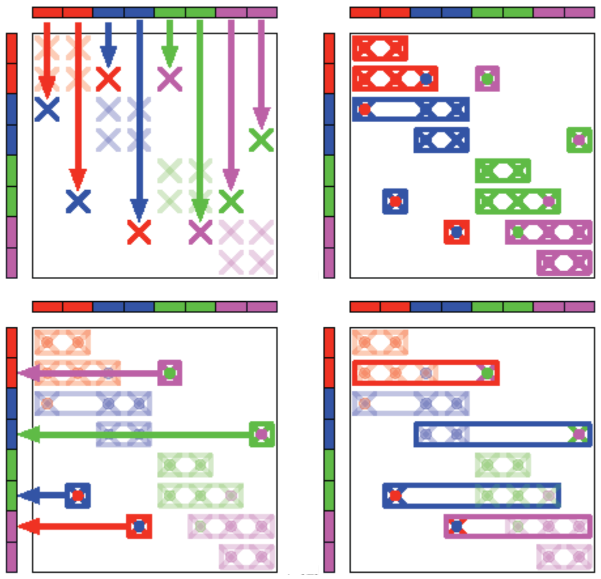
# CSRI SUMMER PROCEEDINGS 2007

The **C**omputer **S**cience **R**esearch **I**nstitute
at Sandia National Laboratories

**Editors:**

Michael L. Parks and S. Scott Collis
Sandia National Laboratories

December 6, 2007



SAND2007-7977

## Preface

The Computer Science Research Institute (CSRI) brings university faculty and students to Sandia National Laboratories for focused collaborative research on computer science, computational science and mathematics problems that are critical to the mission of the laboratories, the Department of Energy, and the United States. CSRI provides a mechanism by which university researchers learn about and impact national- and global-scale problems while simultaneously bringing new ideas from the academic research community to bear on these important problems.

A key component of CSRI programs over the last decade has been an active and productive summer program were students from around the country conduct internships at CSRI. Each student is paired with a Sandia staff member who serves as technical advisor and mentor. The goals of the summer program are to expose the students to research in mathematical and computer sciences at Sandia and to conduct a meaningful and impactful summer research project with their Sandia mentor. Every effort is made to align summer projects with the student's research objectives and all work is coordinated with the ongoing research activities of the Sandia mentor in alignment with Sandia technical thrusts and the needs of the NNSA Advanced Scientific Computing (ASC) program that has funded CSRI from its onset.

Starting in 2006, CSRI has encouraged all summer participants and their mentors to contribute a technical article to the CSRI Summer Proceedings of which this document is the second installment. In many cases, the CSRI proceedings is the first opportunity that students have to write a research article. Not only do these proceedings serve to document the research conducted at CSRI but, as part of the research training goals of CSRI, it is the intent that these articles serve as precursors-to or first-drafts-of articles that could be submitted to peer-reviewed journals. As such, each article has been reviewed by a Sandia staff member knowledgeable in that technical area with feedback provided to the authors. Several articles have or are in the process of being submitted to peer-reviewed conferences or journals and we anticipate that additional submissions will be forthcoming.

For the 2007 CSRI Proceedings, research articles have been organized into the following broad technical focus areas – *computational mathematics and algorithms, discrete mathematics and informatics, transformation, architecture and systems software, and applications* – which are well aligned with Sandia strategic thrusts in computer and information sciences.

We would like to thank all participants who have contributed to the outstanding technical accomplishments of CSRI in 2007 as documented by the high quality articles in this proceedings. The success of CSRI hinged on the hard work of 27 enthusiastic student collaborators and their dedicated Sandia technical staff mentors. It is truly impressive that the research described herein occurred primarily over a three month period of intensive collaboration.

CSRI benefited from the administrative help of Deanna Ceballos, Bernadette Watts, Mel Loran, Dee Cadena, and Vonda Coleman. The success of CSRI is, in large part, due to their dedication and care and it is much appreciated. We would also like to thank those that reviewed articles for this proceedings – their feedback is an important part of the research training process and has significantly improved the quality of the papers herein. We would like to thank David Womble for his advice, guidance and overall CSRI management. Finally, we want to acknowledge the ASC program for their continued support of the CSRI and its activities which have benefited both Sandia and the greater research community.

Michael L. Parks
S. Scott Collis

December 6, 2007

## Table of Contents

## Computational Mathematics and Algorithms

Scientific computation has reached the point where it is on a par with laboratory experiment and mathematical theory as a tool for research in science and engineering and simulation-based engineering science has been declared indispensable to the nation's continued leadership in science and engineering. Indeed, a erudite quotation regarding algorithm development states "I'd rather use today's algorithms on yesterday's computers than the other way around." The fundamental difficulties in modeling faced today are not the kind that can be solved merely by building bigger and faster computers, but also require the development of new computational mathematics.

To that end, the articles in this section focus on fundamental algorithms with broad application. Bochev *et al.* consider an algebraic reformulation of the discrete second-order elliptic equations along with a new algebraic multigrid technique for the fast solution of the reformulated problem. Schroder *et al.* present a new strength of connection criteria for automatic construction of a grid hierarchy in algebraic multigrid. In particular, their criteria is applicable for situations where classical strength measures are ineffective. Nong and Thornquist investigate model-order-reduction techniques in the context of large scale circuit simulation in Xyce. Lieberman and van Bloemen Waanders utilize Hessian-based model reduction to achieve real-time solution of the source inversion problem arising when identifying a contamination source given only sparse sensor information. Karlin and Hu discuss the implementation and profiling of a variable block-matrix multiply within the ML package of algebraic multigrid preconditioners within Trilinos. Davis and Lehoucq explain the Craig-Bampton method for component mode synthesis. Finally, Baker and Lehoucq propose an improved algorithm for the numerical solution of symmetric eigenvalue problems with constraints.

M.L. Parks
S.S. Collis

December 6, 2007

# COMPATIBLE GAUGE APPROACHES FOR *H*(div) EQUATIONS

PAVEL B. BOCHEV[*], CHRISTOPHER M. SIEFERT[*], RAYMOND S. TUMINARO[*], JINCHAO XU[†], AND YUNRONG ZHU[†]

**Abstract.** We are concerned with the compatible gauge reformulation for $H(\text{div})$ equations and the design of fast solvers of the resulting linear algebraic systems as in [5]. We propose an algebraic reformulation of the discrete $H(\text{div})$ equations along with an algebraic multigrid (AMG) technique for the reformulated problem. The reformulation uses discrete Hodge decompositions on co-chains to replace the discrete $H(\text{div})$ equations by an equivalent $2 \times 2$ block linear system whose diagonal blocks are discrete Hodge Laplace operators acting on 2-cochains and 1-cochains respectively. We illustrate the new technique, using the lowest order Raviart-Thomas elements on structured tetrahedral mesh in three dimension and present computational results.

**1. Introduction.** In this paper, we consider general second order elliptic operators over the Lipschitz polyhedral domain $\Omega$ in 3D. Specifically, let $\Omega$ be a bounded, simply connected, and contractible domain in $\mathbb{R}^3$ with Lipschitz boundary $\partial\Omega$. We are looking at the compatible discretization of the following model equation:

$$\left\{ \begin{array}{rcll} -\nabla(\lambda\nabla\cdot u) + \frac{1}{\mu}u &=& f & \text{in } \Omega, \\ \frac{1}{\mu}u\cdot n &=& 0 & \text{on } \Gamma, \\ \lambda\nabla\cdot u &=& 0 & \text{on } \Gamma^*, \end{array} \right. \tag{1.1}$$

where $\partial\Omega = \Gamma \cup \Gamma^*$ and $\Gamma \cap \Gamma^* = \emptyset$. Here, we assume that $\lambda$ and $\mu$ are positive throughout the domain, but may possibly vary widely.

The variational formulation of problem (1.1) leads naturally to the Hilbert space $H(\text{div})$ given by

$$H(\text{div}) := \left\{ u \in \left(L^2(\Omega)\right)^3 \mid \nabla\cdot u \in L^2(\Omega) \right\}.$$

This equation is ubiquitous in problems arising in fluid and solid mechanics [6, 10]. It occurs, in particular, in the solution of second order elliptic partial differential equations (PDE) by first order least-squares methods or by mixed methods with augmented Lagrangians, see [1, 11, 18, 19] and the references cited therein. The importance of $H(\text{div})$-related problems has prompted vigorous research into efficient multilevel schemes, see [1, 11, 12, 18, 19].

The method to be developed in the current paper follows closely the idea of the recent work of Bochev, Hu, Siefert and Tuminaro [5] for Maxwell's equations. Specifically, we propose an algebraic reformulation of the discrete $H(\text{div})$ equations along with a new AMG technique for this reformulated problem. The reformulation process takes advantage of a discrete Hodge decomposition on co-chains to replace the discrete $H(\text{div})$ equations by an equivalent $2 \times 2$ block linear system whose diagonal blocks are discrete Hodge Laplace operators acting on 2-cochains and 1-cochains, respectively. The new AMG algorithm in this paper makes use of the Hiptmair smoother [11] on the fine mesh, uses the canonical interpolations $\Pi_h^{\text{div}}$ and $\Pi_h^{\text{curl}}$ on $H(\text{div})$ and $H(\text{curl})$ to construct the grid-transfer operators, and then uses the standard AMG methods for Laplace-type problems on the coarse meshes.

The rest of the paper is organized as follows. Section 2 reviews basic facts about the discretization framework used in the paper. In Section 3, we apply this framework to obtain a compatible discretization for the $H(\text{div})$ equations and its equivalent reformulation. AMG solvers for the reformulated system are developed in Section 4. In Section 5 we present

[*]Sandia National Laboratories, Computational Math & Algorithms, {pbboche, csiefer, rstumin}@sandia.gov
[†]Penn. State University for Department of Mathematics, {xu, zhu_y}@math.psu.edu

computational results in three dimension that illustrate the new technique in the context of smoothed aggregation AMG. In all experiments we use finite element discretizations based on the lowest order Raviart-Thomas element and lowest order Nédélec element on structured tetrahedral elements.

**2. Compatible discretization framework.** In this section, we give a short introduction of a general framework for compatible discretizations developed in [3]. This framework is based on algebraic topology and includes certain finite element [4, 17], finite volume [15], and finite difference [16] schemes as particular cases. As a result, the AMG algorithm developed in this paper is readily applicable to discrete problems generated by any of these schemes. The presentation here is almost the same as [5, Section3]. We include this section just for the sake of completeness.

**2.1. Computational grid.** We consider computational grids $\Omega^h$ consisting of 0-cells (nodes), 1-cells (edges), 2-cells (faces), and 3-cells (volumes). Formal linear combinations of $k$-cells are called $k$-chains [8]. The sets of $k$-chains forming $\Omega^h$ are denoted by $C_k$. We will assume that $\Omega^h$ is such that the collection $\{C_0, C_1, C_2, C_3\}$ is a complex, i.e., for any $c \in C_k$, $\partial_k c \in C_{k-1}$, where $\partial_k : C_k \mapsto C_{k-1}$ is the boundary operator on $k$-chains [7]. Together with the identity $\partial_k \partial_{k+1} = 0$ this gives rise to the exact sequence

$$0 \longleftarrow C_0 \xleftarrow{\partial_1} C_1 \xleftarrow{\partial_2} C_2 \xleftarrow{\partial_3} C_3 \longleftarrow 0\,. \tag{2.1}$$

The dual of $C_k$ is denoted by $C^k$ and its members are called $k$-cochains [8]. While $C_k$ and $C^k$ are isomorphic, they have different meanings in our discretization framework. The sets $C_k$ represent the physical objects that form the grid, while $C^k$ are collections of real numbers associated with the grid objects. For example, $c_1 \in C_1$ is a formal sum of (oriented) grid edges, while its isomorphic image $c^1 \in C^1$ is a set of real numbers[1] assigned to the edges of $c_1$.

Therefore, the elements of $C^0$ provide values associated with the 0-cells (grid nodes); the elements of $C^1$ are values associated with the 1-cells (grid edges); $C^2$ contains values assigned to the 2-cells (grid faces) of the grid, and $C^3$ are the values assigned to the 3-cells (grid volumes). We will use $C^0$ and $C^3$ to approximate scalar functions and $C^1$ and $C^2$ - to approximate vector functions.

The symbols $C^k_\Gamma$ will denote the subspaces of $C^k$ constrained by zero on the Dirichlet boundary $\Gamma$ for $k = 0, 1, 2$. Such spaces are needed to approximate scalar and vector functions subject to appropriate boundary conditions[2].

**2.2. Natural operators.** Let $\langle\cdot,\cdot\rangle$ denote the duality pairing of $C_k$ and $C^k$. The adjoint of $\partial_k$, defined by $\langle a, \partial_k c\rangle = \langle \delta_k a, c\rangle$, induces an operator $\delta_k : C^k_\Gamma \mapsto C^{k+1}_\Gamma$ called the coboundary. This operator satisfies $\delta_{k+1}\delta_k = 0$ and gives rise to the exact sequence

$$\mathbb{R} \longrightarrow C^0_\Gamma \xrightarrow{\delta_0} C^1_\Gamma \xrightarrow{\delta_1} C^2_\Gamma \xrightarrow{\delta_2} C^3 \longrightarrow 0\,. \tag{2.2}$$

It is not hard to see that the matrix representation $\mathbb{D}_k$ of $\delta_k$ is the signed incidence matrix between $C^k$ and $C^{k+1}$. Following [14] we call $\mathbb{D}_0$, $\mathbb{D}_1$, and $\mathbb{D}_2$ *natural* approximations of the gradient, curl and divergence operators. Note that from $\delta_{k+1}\delta_k = 0$ it follows that

$$\mathbb{D}_{k+1}\mathbb{D}_k = 0;\ k = 0, 1, 2\,, \tag{2.3}$$

[1]Clearly, $C^k$ are isomorphic to $\mathbf{R}^{\tilde{k}}$, where $\tilde{k} = \dim C^k$. For simplicity, the isomorphic image of the cochain $c^k \in C^k$ in $\mathbf{R}^{\tilde{k}}$ will be denoted by the same symbol.

[2]For example, $C^0_\Gamma$ approximates scalar functions such that $\phi = 0$ on $\Gamma$; $C^1_\Gamma$ can be used to approximate vector fields $\mathbf{E}$ such that $\mathbf{n} \times \mathbf{E} = 0$ on $\Gamma$. The space $C^2_\Gamma$ is appropriate for vector fields $\mathbf{B}$ that have a vanishing normal component on $\Gamma$.

and so our natural operators mimic the well-known vector calculus identities $\nabla\times\nabla = 0$, and $\nabla\cdot\nabla\times = 0$. In [13], it is pointed out that natural operations are not enough to provide compatible discretizations of the basic second order operators because their ranges and domains do not match. For example, we cannot approximate $\nabla\times\nabla\times$ by $\mathbb{D}_1\mathbb{D}_1$ because $\mathbb{D}_1$ is in general a rectangular matrix. The number of its columns and rows equals the number of 1-cells and 2-cells in the grid, which are not the same.

**2.3. Metric structures and derived operators.** Let $\mathbb{M}_k : C^k_\Gamma \mapsto C^k_\Gamma$; $k = 0,1,2,3$ denote symmetric positive definite matrices. The matrix $\mathbb{M}_k$ endows $C^k_\Gamma$ with an inner product structure,

$$(a^k, b^k)_{C^k} = (a^k)^T \mathbb{M}_k(b^k)\,. \tag{2.4}$$

The matrices $\mathbb{M}_0$ and $\mathbb{M}_3$ approximate weighted $L^2$ inner products of scalar functions:

$$\mathbb{M}_0 \longrightarrow \int_\Omega \gamma p\hat{p}\, d\Omega\,; \quad \mathbb{M}_3 \longrightarrow \int_\Omega \lambda\phi\hat{\phi}\, d\Omega\,,$$

while $\mathbb{M}_1$ and $\mathbb{M}_2$ approximate the weighted $L^2$ inner products of vector functions

$$\mathbb{M}_1 \longrightarrow \int_\Omega \sigma\mathbf{E}\hat{\mathbf{E}}\, d\Omega\,; \quad \mathbb{M}_2 \longrightarrow \int_\Omega \mu^{-1}\mathbf{B}\hat{\mathbf{B}}\, d\Omega\,.$$

We will also use the notation $\mathbb{M}_0(\gamma)$, $\mathbb{M}_1(\sigma)$, $\mathbb{M}_2(\mu^{-1})$ and $\mathbb{M}_3(\lambda)$ to show the dependency of the coefficients of these mass matrices explicitly.

We define the derived operator $\mathbb{D}^*_k : C^{k+1}_\Gamma \mapsto C^k_\Gamma$ as the adjoint of $\mathbb{D}_k$ with respect to the inner product (2.4):

$$(\mathbb{D}^*_k a^{k+1}, b^k)_{C^k} = (a^{k+1}, \mathbb{D}_k b^k)_{C^{k+1}}\,. \tag{2.5}$$

From (2.5) it is easy to see that for $k = 0,1,2$

$$\mathbb{D}^*_k = \mathbb{M}_k^{-1}\mathbb{D}_k^T\mathbb{M}_{k+1}\,. \tag{2.6}$$

The matrices $\mathbb{D}^*_2$, $\mathbb{D}^*_1$ and $\mathbb{D}^*_0$ provide a second set of discrete differential operators. Specifically, they are approximations of scaled gradient, curl and divergence operators

$$\mathbb{D}^*_2 \to -\mu\nabla\lambda\,; \quad \mathbb{D}^*_1 \to \sigma^{-1}\nabla\times\mu^{-1}\,; \quad \mathbb{D}^*_0 \to -\gamma^{-1}\nabla\cdot\sigma\,,$$

augmented with the boundary conditions

$$\lambda\phi = 0\,; \quad \mathbf{n}\times\mu^{-1}\mathbf{B} = 0\,; \quad \text{and} \quad \mathbf{n}\cdot\sigma\mathbf{E} = 0 \quad \text{on } \Gamma^*\,,$$

respectively. Using (2.6) and (2.3)

$$\mathbb{D}^*_k\mathbb{D}^*_{k+1} = \mathbb{M}_k^{-1}\mathbb{D}_k^T\mathbb{M}_{k+1}\mathbb{M}_{k+1}^{-1}\mathbb{D}_{k+1}^T\mathbb{M}_{k+2} = \mathbb{M}_k^{-1}\mathbb{D}_k^T\mathbb{D}_{k+1}^T\mathbb{M}_{k+2} = 0\,,$$

and so, the basic vector calculus identities hold for the derived operators as well. The commuting diagram, and the relationships among the operators defined above can be summarized in Figure 2.1. Here, the operators $\Pi_h^{\text{grad}}$, $\Pi_h^{\text{curl}}$, $\Pi_h^{\text{div}}$, and $\Pi_h^0$ are the canonical interpolations on $H^1(\Omega)$, $H(\text{curl})$, $H(\text{div})$, and $L^2(\Omega)$ to the corresponding finite element spaces $V_h(\text{grad})$, $V_h(\text{curl})$, $V_h(\text{div})$, and $V_h(0)$ respectively. The lower half of the commuting diagram above presents the relationships among the operators. For example, from this diagram we can easily find out that

$$\mathbb{D}^*_2 = \mathbb{M}_2^{-1}\mathbb{D}_2^T\mathbb{M}_3.$$

$$\begin{tikzcd}
\mathbb{R} \arrow[r, "I"] & H^1(\Omega) \arrow[r, "\nabla"] \arrow[d, "\Pi_h^{\mathrm{grad}}"] & H(\mathrm{curl}) \arrow[r, "\nabla\times"] \arrow[d, "\Pi_h^{\mathrm{curl}}"] & H(\mathrm{div}) \arrow[r, "\nabla\cdot"] \arrow[d, "\Pi_h^{\mathrm{div}}"] & L^2(\Omega) \arrow[r] \arrow[d, "\Pi_h^{0}"] & 0 \\
\mathbb{R} \arrow[r, "I"] & V_h(\mathrm{grad}) \arrow[r, "\mathbb{D}_0", "\mathbb{D}_0^T"'] \arrow[d, "\mathbb{M}_0(\gamma)"] & V_h(\mathrm{curl}) \arrow[r, "\mathbb{D}_1", "\mathbb{D}_1^T"'] \arrow[d, "\mathbb{M}_1(\sigma)"] & V_h(\mathrm{div}) \arrow[r, "\mathbb{D}_2", "\mathbb{D}_2^T"'] \arrow[d, "\mathbb{M}_2(\mu^{-1})"] & V_h(0) \arrow[r] \arrow[d, "\mathbb{M}_3(\lambda)"] & 0 \\
\mathbb{R} & V_h(\mathrm{grad}) \arrow[l, "I"'] & V_h(\mathrm{curl}) \arrow[l, "\mathbb{D}_0^*"'] & V_h(\mathrm{div}) \arrow[l, "\mathbb{D}_1^*"'] & V_h(0) \arrow[l, "\mathbb{D}_2^*"'] & 0 \arrow[l]
\end{tikzcd}$$

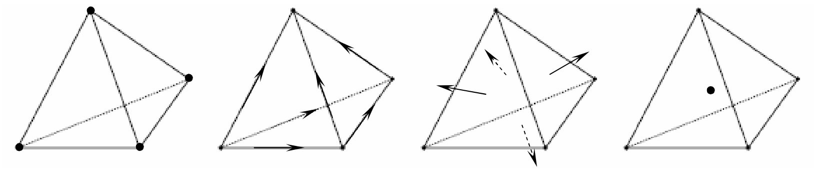


FIG. 2.1. *De Rahm Complex and the lowest order finite element spaces*

Because the range of $\mathbb{D}_k$ is contained in the domain of $\mathbb{D}_k^*$ and vice versa we can use the natural and the derived operators to define discrete versions of the basic second order differential operators, including a discrete Hodge Laplace operator. Specifically, for $k = 0, 1, 2$ we have the second order operators

$$\mathbb{D}_k^*\mathbb{D}_k = \mathbb{M}_k^{-1}\mathbb{D}_k^T\mathbb{M}_{k+1}\mathbb{D}_k \ : C_\Gamma^k \mapsto C_\Gamma^k \tag{2.7}$$

$$\mathbb{D}_k\mathbb{D}_k^* = \mathbb{D}_k\mathbb{M}_k^{-1}\mathbb{D}_k^T\mathbb{M}_{k+1} \ : C_\Gamma^{k+1} \mapsto C_\Gamma^{k+1} \tag{2.8}$$

and the discrete Hodge Laplacian

$$\mathbb{L}_k : C_\Gamma^k \mapsto C_\Gamma^k; \quad \mathbb{L}_k = \mathbb{D}_k^*\mathbb{D}_k + \mathbb{D}_{k-1}\mathbb{D}_{k-1}^*; \quad k = 0, 1, 2, 3 \tag{2.9}$$

with the understanding that $\mathbb{D}_3 = 0$ and $\mathbb{D}_{-1}^* = 0$.

The discrete operators in (2.7)-(2.9) approximate basic second order elliptic differential operators. In §3.1 we will use these operators to motivate and explain our reformulation strategy.

Similar to [5], we also introduce a second set of inner products defined by the matrices $\widetilde{\mathbb{M}}_k$, $(k = 0, 1, 2, 3)$ that uses a unit weight, i.e.,

$$\widetilde{\mathbb{M}}_k \to \int_\Omega u^k v^k d\Omega, \quad u^k, v^k \in C_\Gamma^k.$$

These inner products can be used to define a second set of derived operators $\widetilde{\mathbb{D}}_k^* : C_\Gamma^{k+1} \mapsto \widetilde{\mathbb{D}}_k^*$ given

$$\widetilde{\mathbb{D}}_k^* = \mathbb{M}_k^{-1}\mathbb{D}_k^T\widetilde{\mathbb{M}}_{k+1}, k = 0, 1, 2$$

respectively, and such that $\widetilde{\mathbb{D}}_k^*\widetilde{\mathbb{D}}_{k+1}^* = 0$. These operators give rise to the discrete Hodge Laplace operators

$$\widetilde{\mathbb{L}}_k : C_\Gamma^k \mapsto C_\Gamma^k; \qquad \widetilde{\mathbb{L}}_k = \widetilde{\mathbb{D}}_k^*\mathbb{D}_k;$$

that are different versions of $\mathbb{L}_k$ respectively.

The following general result from [3] provides the results needed for the reformulation of the discrete $H(\mathrm{div})$ equation.

**Theorem 2.1** *The size of the kernel of the analytic and discrete Hodge Laplacians is the same.*

Theorem 2.1 reveals that the null-space of the discrete Hodge Laplacian and, by extension the structure of the discrete Hodge decomposition of discrete functions in $C^k_\Gamma$, are topological invariants that are independent of the particular choice of metric, i.e., the matrices $\mathbb{M}_k$. As a result, the assertion of this theorem is valid for both $\mathbb{L}_0$, $\mathbb{L}_1$, $\mathbb{L}_2$, and $\widetilde{\mathbb{L}}_0$, $\widetilde{\mathbb{L}}_1$, $\widetilde{\mathbb{L}}_2$. The properties of these operators, relevant to the reformulation process, are summarized in the following corollary, which is a generalization of [5, Corollary 3.2].

**Corollary 2.2** *Assume that $\Omega$ is contractible. Then, every $u^k \in C^k_\Gamma$ $(k = 1, 2)$ has the discrete Hodge decomposition*

$$u^k = \mathbb{D}_{k-1} p^{k-1} + \widetilde{\mathbb{D}}^*_k b^{k+1} \tag{2.10}$$

*where $p^{k-1} \in C^{k-1}_\Gamma$ and $b^{k+1} \in C^{k+1}_\Gamma$ solve the equations*

$$\widetilde{\mathbb{D}}^*_{k-1}\mathbb{D}_{k-1} p^{k-1} = \widetilde{\mathbb{D}}^*_{k-1} u^k \quad \textit{and} \quad \mathbb{D}_k \widetilde{\mathbb{D}}^*_k b^{k+1} = \mathbb{D}_k u^k\,, \tag{2.11}$$

*respectively.*

**3. Compatible discretization of $H(\mathrm{div})$ equation.** Using the discrete operators defined in the last section, a compatible discretization of the $H(\mathrm{div})$ equation (1.1) is straightforward. Specifically, we approximate $u$ by a 2-cochain $u^2 \in C^2_\Gamma$ that is associated with the 2-cells (the faces) of the mesh that are not in $\Gamma$. Then the compatible discrete version of the $\nabla\nabla\cdot$ operator is provided by the second order discrete operator $\mathbb{D}^*_2\mathbb{D}_2$. As a result, the compatible, fully discrete equation of (1.1) is given by

$$(\mathbb{D}_2^T \mathbb{M}_3 \mathbb{D}_2 + \mathbb{M}_2) u^2 = f^2, \tag{3.1}$$

with the matrix $\mathbb{M}_3$ containing the material parameter $\lambda$ and the matrix $\mathbb{M}_2$ containing $\mu^{-1}$ and $f^2 \in C^2_\Gamma$ is a discrete version of $f$ in (1.1). An equivalent "weak" form of (3.1) is given by the variational equation: seek $u^2 \in C^2_\Gamma$ such that

$$\left(u^2, \hat{u}^2\right)_{C^2} + \left(\mathbb{D}_2 u^2, \mathbb{D}_2 \hat{u}^2\right)_{C^3} = \left(f^2, \hat{u}^2\right) \quad \forall \hat{u}^2 \in C^2_\Gamma\,. \tag{3.2}$$

**3.1. Reformulation.** Following [5] for Maxwell's equations, we intend on forming the Hodge Laplacian, which here corresponds to adding a $\nabla\times\nabla\times$ term, namely

$$\mathbb{L}_2 = \mathbb{D}^*_2\mathbb{D}_2 + \mathbb{D}_1\mathbb{D}^*_1. \tag{3.3}$$

The following main theorem states an analogue of Theorem 4.2 in [5].

**Theorem 3.1** *Assume that $u^2$ is a solution of* (3.1) *and let*

$$u^2 = \mathbb{D}_1 e^1 + \widetilde{\mathbb{D}}^*_2 b^3 \tag{3.4}$$

*denote its discrete Hodge decomposition with respect to the inner product induced by $\widetilde{\mathbb{M}}_2$. The pair $(a^2, e^1)$, where $a^2 = \widetilde{\mathbb{D}}^*_2 b^3$, solves the linear system*

$$\begin{bmatrix} \mathbb{M}_2 + \mathbb{D}_2^T\mathbb{M}_3\mathbb{D}_2 + \widetilde{\mathbb{M}}_2\mathbb{D}_1\mathbb{M}_1^{-1}\mathbb{D}_1^T\widetilde{\mathbb{M}}_2 & \mathbb{M}_2\mathbb{D}_1 \\ \mathbb{D}_1^T\mathbb{M}_2 & \mathbb{D}_1^T\mathbb{M}_2\mathbb{D}_1 \end{bmatrix} \begin{bmatrix} a^2 \\ e^1 \end{bmatrix} = \begin{bmatrix} \mathbb{M}_2 f^2 \\ \mathbb{D}_1^T\mathbb{M}_2 f^2 \end{bmatrix}. \tag{3.5}$$

*Proof.* Denoting $a^2 = \widetilde{\mathbb{D}}_2^* b^3$, and applying the decomposition (3.4) to the weak form (3.2) gives

$$\left(\mathbb{D}_1 e^1 + a^2, \hat{u}^2\right)_{C^2_\Gamma} + \left(\mathbb{D}_2 a^2, \mathbb{D}_2 \hat{u}^2\right)_{C^3} = \left(f^2, \hat{u}^2\right)_{C^2_\Gamma}, \ \forall \hat{u}^2 \in C^2_\Gamma.$$

In the above equality, we used the fact that $\mathbb{D}_2\mathbb{D}_1 \equiv 0$. We note that the assumed Hodge decomposition implies that $\widetilde{\mathbb{D}}_1^* a^2 = 0$ (since $\widetilde{\mathbb{D}}_1^*\widetilde{\mathbb{D}}_2^* = 0$), thus

$$\left(\widetilde{\mathbb{D}}_1^* a^2, \widetilde{\mathbb{D}}_1^* \hat{u}^2\right)_{C^1_\Gamma} \equiv 0, \quad \forall \hat{u}^2 \in C^2_\Gamma.$$

As a result, this term can be added to the last equation to obtain:

$$\left(\mathbb{D}_1 e^1 + a^2, \hat{u}^2\right)_{C^2_\Gamma} + \left(\mathbb{D}_2 a^2, \mathbb{D}_2 \hat{u}^2\right)_{C^3} + \left(\widetilde{\mathbb{D}}_1^* a^2, \widetilde{\mathbb{D}}_1^* \hat{u}^2\right)_{C^1_\Gamma} = \left(f^2, \hat{u}^2\right)_{C^2_\Gamma}, \ \forall \hat{u}^2 \in C^2_\Gamma.$$

It is easy to see that the above weak form is equivalent to the following linear system:

$$\mathbb{M}_2 a^2 + \left(\mathbb{D}_2^T \mathbb{M}_3 \mathbb{D}_2 + \widetilde{\mathbb{M}}_2 \mathbb{D}_1 \mathbb{M}_1^{-1} \mathbb{D}_1^T \widetilde{\mathbb{M}}_2\right) + \mathbb{M}_2 \mathbb{D}_1 e^1 = \mathbb{M}_2 f^2$$

which is the first equation in (3.5).

Applying the decomposition (3.4) to (3.1), and then multiplying by $\mathbb{D}_1^*$ on both sides gives

$$\mathbb{D}_1^* a^2 + \mathbb{D}_1^* \mathbb{D}_1 e^1 = \mathbb{D}_1^* f^2.$$

Noticing that by definition $\mathbb{D}_1^* = \mathbb{M}_1^{-1}\mathbb{D}_1^T\mathbb{M}_2$, the second set of equations in the block system follows by multiplying $\mathbb{M}_1$ on both sides. This completes the proof. □

Here, we should notice that the (2,2) block $\mathbb{D}_1^*\mathbb{D}_1$ is singular. A further decomposition [5, Corollary 3.2] of

$$e^1 = \mathbb{D}_0 e^0 + \widetilde{\mathbb{D}}_1^* b^2 := \mathbb{D}_0 e^0 + a^1$$

yields the following block system

$$\begin{bmatrix} A_{11} & \mathbb{M}_2\mathbb{D}_1 \\ \mathbb{D}_1^T\mathbb{M}_2 & A_{22} \end{bmatrix} \begin{bmatrix} a^2 \\ a^1 \end{bmatrix} = \begin{bmatrix} \mathbb{M}_2 f^2 \\ \mathbb{D}_1^T \mathbb{M}_2 f^2 \end{bmatrix} \tag{3.6}$$

where $A_{11} = \mathbb{M}_2 + \mathbb{D}_2^T\mathbb{M}_3\mathbb{D}_2 + \widetilde{\mathbb{M}}_2\mathbb{D}_1\mathbb{M}_1^{-1}\mathbb{D}_1^T\widetilde{\mathbb{M}}_2$ and $A_{22} = \mathbb{D}_1^T\mathbb{M}_2\mathbb{D}_1 + \widetilde{\mathbb{M}}_1\mathbb{D}_0\mathbb{M}_0^{-1}\mathbb{D}_0\widetilde{\mathbb{M}}_1$. In the above formulation, we used the fact that $\mathbb{D}_1\mathbb{D}_0 = 0$ and $\widetilde{\mathbb{D}}_0^*\widetilde{\mathbb{D}}_1^* = 0$.

**Remark 3.2** *The reformulation* (3.6) *seems more complicated than the original equation* (3.1) *that we are actually solving. The idea here is try to use the diagonal blocks $A_{11}$ and $A_{22}$ as preconditioner of* (3.1)*, which is the main focus of the next section.*

*It is interesting to notice that during this reformulation procedure, the gauge term in the $A_{11}$ and $A_{22}$ blocks seems to be indispensable. As was pointed out in [5], these terms play an important role in avoiding the large null-space caused by $\nabla\nabla\cdot$ operator and $\nabla\times\nabla\times$ operator respectively. Forming $\widetilde{\mathbb{M}}_2\mathbb{D}_1\mathbb{M}_1^{-1}\mathbb{D}_1^T\widetilde{\mathbb{M}}_2$ and $\widetilde{\mathbb{M}}_1\mathbb{D}_0\mathbb{M}_0^{-1}\mathbb{D}_0\widetilde{\mathbb{M}}_1$ requires the inversion of $\mathbb{M}_1$ and $\mathbb{M}_0$. Even if we can use mass lumping to simplify the computation, it makes the system more complicated and ruins the sparsity pattern of the original system. The interesting fact is that according to the numerical tests (see Section 5 for more details), it is not so clear now if these gauge terms are necessary or not. We need a more rigorous investigation of the roles of these gauge terms for more complex problems.*

**4. Multigrid solvers.** Now we are in position to combine the reformulation and preconditioning to develop a linear solver for the compatible discretization (3.1) of the $H(\text{div})$ equation (1.1). Similar to the algorithm in [5], we focus on developing the AMG block preconditioners.

The approach considered in this paper focuses on developing AMG methods for the (1,1) and (2,2) blocks in (3.6) separately. Note that these diagonal blocks are Laplace-like. Once constructed, these AMG solvers are combined as a Jacobi-like preconditioner to precondition (1.1).

We propose an AMG technique for the whole $2 \times 2$ system which employs a Hiptmair smoother (see for example [11]) at the finest level, but allows subsequent levels and transfers of the (1,1) and (2,2) blocks to be handled with the standard AMG method. To do this, the face element of the (1,1) block and the edge element version of the (2,2) block must be converted to a more standard nodal form on the coarse mesh. This is accomplished by two special prolongators that not only transfer solutions from a coarse to a fine solution but also transfer solutions from a nodal to a face or edge representation, respectively. The net effect of these special prolongators is that the corresponding Galerkin projection of the (1,1) and (2,2) block will, in fact, yield a coarse operator resembling a vector nodal Laplacian which is amenable to any standard AMG method for further coarsening.

**4.1. The specialized prolongators.** As discussed earlier, in order to use the standard AMG solvers for the (1,1) and (2,2) block, we must convert the face element (for the (1,1)-block) and the edge element (for the (2,2)-block) into the standard nodal form. To do this, we define specialized prolongators $P_{11}$ and $P_{22}$ to transfer solutions from a nodal to a face and edge representation respectively. Instead of introducing the near null-space to define the prolongators as was done in [5], here we make use of the interpolation $\Pi_h^{\text{div}}$ and $\Pi_h^{\text{curl}}$ (see Figure 2.1) as in [2] and [12].

There are many ways to obtain aggregates corresponding to nodes, see [5] for more details. In this paper, for simplicity we use perfect aggregation. By "perfect", we mean that the aggregates are formed manually. Note that we only need to form these aggregates on the finest level. Once the aggregates are formed, $\Pi_h^{\text{div}}$ and $\Pi_h^{\text{curl}}$ must also be computed. The detailed construction of the special prolongators for the (1,1) and (2,2) block is given in Algorithm 1. Notice that the net effect of $P_{11}$ is to interpolate coarse nodal quantities to fine face-oriented quantities, and the effect of $P_{22}$ is to interpolate coarse nodal quantities to fine edge-oriented quantities.

**Algorithm 1**: $[P_{11}, P_{22}]$=Coarse_Node_Prolongators()

```
1 $\{\mathcal{A}_i\}$ ←Aggregate manually;
2 For each fine node $n_i$ and each aggregate $\mathcal{A}_j$ define
      $(P_{nf})_{i,j} = \begin{cases} 1, & \text{if } n_i \in \mathcal{A}_j \\ 0, & \text{otherwise} \end{cases}$ .
3 $P_{11} = \Pi_h^{\text{div}} P_{nf}$;
4 $P_{22} = \Pi_h^{\text{curl}} P_{nf}$;
```

The Galerkin coarse discretizations are given by

$$A_{11}^H = P_{11}^T A_{11} P_{11}, \quad A_{22}^H = P_{22}^T A_{22} P_{22}$$

where $A_{11}$ and $A_{22}$ are the (1,1) and (2,2) block of (3.6), $A_{11}^H$ and $A_{22}^H$ refer to their projections on a coarse mesh, respectively.

**4.2. Relaxation.** As before, we consider the following hybrid scheme. Suppose that the conjugate gradient iteration is actually applied to (3.1) and that (3.6) is *only used within the preconditioner*. To do this, it is necessary to convert residuals of (3.1) to right hand sides of (3.6) within the preconditioner. This is done by applying $[\mathbb{I} \quad \mathbb{D}_1]^T$ to the residual. Approximate solutions to (3.6) are then converted back to a form suitable for (3.1) via $\mathbb{D}_1 a^1 + a^2$.

Algorithm 2 illustrates such a smoother proposed by Hiptmair that combines standard smoothing of the original equations with standard smoothing of the equations projected to the null-space [11].

**Algorithm 2**: $\tilde{u}$ = FineRelaxation($A, \mathbb{D}_1, \tilde{u}, b$)

```
1 $\tilde{u} \leftarrow$ StandardRelaxation($A, \tilde{u}, b$) ;
2 $c \leftarrow$ StandardRelaxation($\mathbb{D}_1^T A \mathbb{D}_1, 0, \mathbb{D}_1^T (b - A\tilde{u})$) ;
3 $\tilde{u} \leftarrow \tilde{u} + \mathbb{D}_1 c$ ;
4 $\tilde{u} \leftarrow$ StandardRelaxation($A, \tilde{u}, b$) ;
```

The key is that the error is smooth after this initial relaxation. Since the error is smooth, fine grid relaxation may be omitted from the AMG V-cycles in Solve(), as (3.1) and (1.1) are equivalent.

It is important to realize that this special smoother is only needed on the finest level. A standard smoother can be used on coarse levels within the AMG procedures for the (1,1) and (2,2) blocks. Finally, an additive version of the Hiptmair smoother may also be considered for FineRelaxation().

**4.3. AMG algorithm preconditioner.** We now give the entire AMG-based preconditioner for the block Jacobi version in Algorithm 3. PreFineRelaxation() is identical to Algorithm 2 except step one is omitted. This also avoids the residual calculation in step two as the initial guess to a preconditioner is always zero. PostFineRelaxation() is identical to Algorithm 2 except step four is omitted to keep the preconditioner symmetric when StandardRelaxation() employs a symmetric algorithm. Of course, residual calculations can also be avoided using additive forms of this smoother.

The algorithm essentially involves two AMG solves for nodal vector Laplacians: $A_{11}^H$ corresponding to the (1,1) block and $A_{22}^H$ corresponding to the (2,2) block. In addition, some relaxation must be performed on the original fine mesh system. Specifically, there are three major components of the preconditioner.

(1) Hiptmair smoother for $H(\text{div})$ (see also Hiptmair [11]).
(2) AMG for $P_{11}^T A_{11} P_{11}$ within the (1,1)-block.
(3) AMG for $P_{22}^T A_{22} P_{22}$ within the (2,2)-block.

The detailed algorithm is listed as follows:

**5. Numerical results.** All the numerical experiments are conducted in a three-dimensional unit cube domain $\Omega = \{(x, y, z) \in \mathbb{R}^3 : 0 \leq x, y, z \leq 1\}$ with homogeneous Neumann boundary condition. The domain is meshed by uniform cubes, and each cube is divided into 6 tetrahedra.

The proposed solver was implemented using CG in MATLAB. The first level and the first grid transfer of Algorithm 3 is also implemented in MATLAB. ML's smoothed aggregation solver is used for $A_{11}^H$ and $A_{22}^H$, through the mlmex MATLAB interface [9]. A single

**Algorithm 3**: $\tilde{u}$ =Block Preconditioner($r$)

```
% Setup Phase
Form $A_{11}^H \leftarrow P_{11}^T A_{11} P_{11}$ efficiently;
Standard_AMG_Setup($A_{11}^H$);
Form $A_{22}^H \leftarrow P_{22}^T A_{22} P_{22}$ efficiently;
Standard_AMG_Setup($A_{22}^H$);
______________________________________________ ;
% Solve Phase
$\tilde{u} \leftarrow$ PreFineRelaxation($\mathbb{D}_2^T \mathbb{M}_3 \mathbb{D}_2 + \mathbb{M}_2, \mathbb{D}_1, 0, r$);
$\tilde{r} \leftarrow r - (\mathbb{D}_2^T \mathbb{M}_3 \mathbb{D}_2 + \mathbb{M}_2)\tilde{u}$;
% Perform V-cycles on $A_{11}^H$ and $A_{22}^H$
$a \leftarrow$ Standard_AMG_Vcycle($A_{11}^H, 0, P_{11}^T \tilde{r}$) ;
$p \leftarrow$ Standard_AMG_Vcycle($A_{22}^H, 0, P_{22}^T \mathbb{D}_1^T \tilde{r}$) ;
$\tilde{u} \leftarrow \tilde{u} + P_{11} a + \mathbb{D}_1 P_{22} p$ ;
$\tilde{u} \leftarrow$ PostFineRelaxation($\mathbb{D}_2^T \mathbb{M}_3 \mathbb{D}_2 + \mathbb{M}_2, \mathbb{D}_1, \tilde{u}, r$) ;
```

Table 5.1
*Number of iterations for CG-accelerated AMG on the 3D tetrahedral mesh problem with constant coefficients, using Algorithm 3. The size of the problem and the number of SGS smoothing steps are varied.*

| Grid | | $9^3$ | $12^3$ | $15^3$ | $18^3$ | $21^3$ | $24^3$ | $27^3$ |
|---|---|---|---|---|---|---|---|---|
| 2 SGS Steps | gauge | 12 | 12 | 13 | 13 | 13 | 13 | 13 |
| | No gauge | 11 | 13 | 13 | 14 | 14 | 14 | 15 |
| 3 SGS Steps | gauge | 10 | 11 | 11 | 12 | 12 | 12 | 12 |
| | No gauge | 9 | 10 | 11 | 12 | 12 | 13 | 13 |
| 4 SGS Steps | gauge | 9 | 10 | 10 | 11 | 11 | 11 | 11 |
| | No gauge | 8 | 10 | 10 | 10 | 11 | 11 | 11 |

V-cycle of AMG is used for both the (1,1) and (2,2) block, using the efficient variant of Algorithm 2 (smoother). Unless otherwise stated, we use two steps of symmetric Gauss-Seidel sub-smoothing on both faces and edges. For all experiments the CG tolerance is $1 \times 10^{-10}$.

**5.1. Constant coefficients.** As the first experiment, we consider the constant coefficients case. We assume that $\lambda = \mu = 1$ in $\Omega$. Table 5.1 reports the number of iterations with different meshsize. We note that the number of iterations are almost identical whether we include the gauge terms in the (1,1), and (2,2)-block or not. By this reason, we will omit the gauge term in the following numerical experiments.

**5.2. Variable $\mu$.** We experiment with jumps in $\mu$ by considering two regions with constant values of $\mu$. Specifically, define

$$\Omega_0 = \left\{(x,y,z) : \frac{1}{3} \le x,y,z \le \frac{2}{3}\right\}, \quad \Omega_1 = \Omega \setminus \Omega_0;$$

let $\mu \equiv 1$ in $\Omega_1$ and choose $\mu = \mu_0$ to be a constant inside $\Omega_0$. $\lambda$ is fixed to be 1 throughout the whole domain $\Omega$. Table 5.2 reports the number of iterations on different meshsize. Note that the number of iterations are quite robust with respect to the variation of the coefficient $\mu$.

Table 5.2
*Number of iterations for CG-accelerated AMG on the 3D tetrahedral mesh problem with jump coefficients, using Algorithm 3.* $\mu_0$ *varies inside* $[1/3, 2/3]^3$*, and 1 elsewhere, and* $\lambda \equiv 1$.

| | $\mu_0^{-1}$ | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Grid | $10^{-8}$ | $10^{-7}$ | $10^{-6}$ | $10^{-5}$ | $10^{-4}$ | $10^{3}$ | $10^{-2}$ | $10^{-1}$ | 1 |
| $9^3$ | 11 | 11 | 11 | 11 | 11 | 11 | 11 | 11 | 11 |
| $18^3$ | 15 | 15 | 15 | 15 | 16 | 16 | 15 | 15 | 14 |
| $27^3$ | 16 | 16 | 19 | 18 | 18 | 18 | 19 | 17 | 15 |

Table 5.3
*Number of iterations for CG-accelerated AMG on the 3D tetrahedral mesh problem with jump coefficients, using Algorithm 3.* $\lambda_0$ *varies inside* $[1/3, 2/3]^3$*, and 1 elsewhere, and* $\mu \equiv 1$.

| | $\lambda_0$ | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Grid | $10^{-4}$ | $10^{-3}$ | $10^{-2}$ | $10^{-1}$ | 1 | $10^{1}$ | $10^{2}$ | $10^{3}$ | $10^{4}$ |
| $9^3$ | 17 | 16 | 14 | 12 | 11 | 11 | 11 | 11 | 9 |
| $18^3$ | 21 | 20 | 18 | 16 | 14 | 14 | 14 | 12 | 12 |
| $27^3$ | 22 | 21 | 21 | 17 | 15 | 15 | 14 | 13 | 13 |

**5.3. Variable $\lambda$.** We now consider the jump on $\lambda$. Same as before, we choose $\lambda = \lambda_0$ to be a constant which varies from $10^{-4}$ to $10^4$ inside the domain $\Omega_0$, and $\lambda = 1$ elsewhere. This time, we fix $\mu$ to be 1 in the whole domain $\Omega$. Table 5.3 reports the number of iterations on different meshsize. Again, the number of iterations remains fairly constant.

**6. Conclusions.** In this paper, we proposed an AMG based preconditioner for the $H(\text{div})$ equation. We reformulated the equation by using the compatible gauge approaches, and formed a $2 \times 2$ system which is equivalent to the original discrete linear equations. Then we combined the AMG solvers for the (1,1) and (2,2) blocks of this system in certain way, and used it as the preconditioner of the original linear system. We also presented some numerical experiments to show the robustness of this algorithm. These experiments showed that the algorithm is very robust even with the presence of large jump coefficients.

## REFERENCES

[1] D. N. Arnold, R. S. Falk, and R. Winther, *Preconditioning in H(div) and applications*, Mathematics of Computation, 66 (1997), pp. 957–984.

[2] R. Beck, *Algebraic multigrid by component splitting for edge elements on simplicial triangulations*, Tech. Report SC 99-40, Zuse Institute Berlin, December 1999.

[3] P. Bochev and J. Hyman, *Principles of mimetic discretizations of differential operators*, in Compatible Spatial Discretizations, D. Arnold, P. Bochev, R. Lehoucq, R. Nicolaides, and M. Shashkov, eds., Springer-Verlag, 2006.

[4] P. Bochev and A. Robinson, *Matching algorithms with physics: exact sequences of finite element spaces*, in Preservation of stability under discretization, D. Estep and S. Tavener, eds., Philadelphia, 2001, SIAM, pp. 145–165.

[5] P. B. Bochev, J. J. Hu, C. M. Siefert, and R. S. Tuminaro, *An algebraic multigrid approach based on a compatible gauge reformulation of Maxwell's equations*, Tech. Report SAND2007-1633J, Sandia National Laboratory, 2007.

[6] F. Brezzi and M. Fortin, *Mixed and Hybrid Finite Element Methods*, vol. 15 of Springer series in computational mathematics, Springer-Verlag, 1991.

[7] S. Cairns, *Introductory topology*, Ronald Press Co., New York, 1961.

[8] A. Dezin, *Multidimensional analysis and discrete models*, CRC Press, Boca Raton, 1995.

[9] M. Gee, C. Siefert, J. Hu, R. Tuminaro, and M. Sala, *ML 5.0 smoothed aggregation user's guide*, Tech. Report SAND2006-2649, Sandia National Laboratories, 2006.

[10] V. Girault and P. A. Raviart, *Finite element methods for Navier-Stokes equations*, Springer-Verlag, Berlin, 1986. Theory and algorithms.

[11] R. Hiptmair, *Multigrid method for $H$(div) in three dimensions*, Electron. Trans. Numer. Anal., 6 (1997), pp. 133–152. Special issue on multilevel methods (Copper Mountain, CO, 1997).

[12] R. Hiptmair and J. Xu, *Nodal auxiliary space preconditioning in H(curl) and H(div) spaces*, tech. report, 2006.

[13] J. Hyman and M. Shashkov, *Adjoint operators for the natural discretizations of the divergence, gradient and curl on logically rectangular grids*, Appl. Num. Math., 25 (1997), pp. 413–442.

[14] ———, *Natural discretizations for the divergence, gradient and curl on logically rectangular grids*, Comput. Math. Appl., 33 (1997), pp. 88–104.

[15] R. Nicolaides, *Direct discretization of planar div-curl problems*, SIAM J. Numer. Anal., 29 (1992), pp. 32–56.

[16] M. Shashkov, *Conservative finite difference methods on general grids*, CRC Press, Boca Raton, FL, 1996.

[17] J. van Welij, *Calculation of eddy currents in terms of H on hexahedra*, IEEE Transactions on Magnetics, 21 (1985), pp. 2239–2241.

[18] P. S. Vassilevski and J. Wang, *Multilevel iterative methods for mixed finite element discretizations of elliptic problems*, Numerische Mathematik, 63 (1992), pp. 503–520.

[19] B. I. Wohlmuth, A. Toselli, and O. B. Widlund, *An iterative substructuring method for Raviart–Thomas vector fields in three dimensions*, SIAM Journal on Numerical Analysis, 37 (2000), pp. 1657–1676.

# GENERALIZED STRENGTH OF CONNECTION IN ALGEBRAIC MULTIGRID

JACOB SCHRODER[‡], RAYMOND S. TUMINARO[§], AND LUKE OLSON[¶]

**Abstract.** Algebraic multigrid (AMG) solves sparse linear systems without knowledge of any underlying geometric grid. The automatic construction of a multigrid hierarchy requires strength of connection information in order to coarsen the matrix graph and determine sparsity patterns for each intergrid transfer operator. This paper focuses on accessing strength of connection information, i.e. determining which degrees of freedom are strongly related to each other when algebraically smooth error is transferred between grids. Unfortunately, classic strength measures based on matrix stencils can be ineffective due to discretization errors and matrix inverses can be too global. We present an ODE framework for interpreting previous measures and propose a new strength of connection criteria. Some numerical results for the new criteria are also given.

**1. Introduction.** Algebraic multigrid (AMG) solves sparse linear systems without knowledge of any underlying geometric grid. The automatic construction of a multigrid hierarchy normally centers on three distinct tasks: coarse grid selection, determination of the sparsity pattern for each intergrid transfer, and the specification of the actual coefficients within intergrid transfer matrices. This paper focuses on the first two tasks which in turn rely on accessing strength of connection information, i.e. determining which degrees of freedom are strongly related to each other when algebraically smooth error is transferred between grids. Specifically, strength of connection information is used to construct a graph, $G$, whose vertices are the degrees of freedom present in the operator, $A$, and where $i$ is connected by an edge to $j$ only if $i$ is strongly connected to $j$. The coarse grid is then constructed by applying some graph algorithm that coarsens $G$. Strength information is also used to construct the intergrid transfer operator, where degree of freedom, $i$, is used to interpolate to degree of freedom, $j$, only if $i$ is strong connected to $j$.

The current state of strength of connection in AMG is primarily based on the seminal work of the 1980's that developed the classic strength of connection measure. The classic strength of connection measure uses the matrix stencil to determine the strength of connection between two degrees of freedom, $i$ and $j$. For instance in [4], $i$ is strongly connected to $j$ with respect to a matrix $A$ only if

$$-A(i,j) \geq \theta \max_{l \neq i}\{-A(i,l)\}, \tag{1.1a}$$

for some drop tolerance, $0 < \theta \quad \leq 1.0$. Similarly, smoothed aggregation [5] sets degree of freedom $i$ to be strongly connected to degree of freedom $j$ only if

$$|A(i,j)| \geq \theta \sqrt{A(i,i)\,A(j,j)}. \tag{1.1b}$$

Unfortunately, the classic measure is most applicable to only M or near-M matrices.

A simple and common example of this measure's limitations can be seen by considering the use of bi-linear finite elements for

$$-u_{xx} + -\epsilon u_{yy} = f \tag{1.2}$$

on a uniform mesh. The corresponding matrix stencil at an interior point is

$$\frac{1}{3}\left(\begin{array}{ccc} -\frac{1+\epsilon}{2} & 1-2\epsilon & -\frac{1+\epsilon}{2} \\ -2+\epsilon & 4+4\epsilon & -2+\epsilon \\ -\frac{1+\epsilon}{2} & 1-2\epsilon & -\frac{1+\epsilon}{2} \end{array}\right). \tag{1.3a}$$

[‡]Department of Computer Science, University of Illinois at Urbana-Champaign, jschrod3@uiuc.edu
[§]Sandia National Laboratories, rstumin@sandia.gov
[¶]Department of Computer Science, University of Illinois at Urbana-Champaign, lukeo@uiuc.edu

$\epsilon = 1.0$ and $\epsilon = 0.0$ yield respectively,

$$\frac{1}{3}\begin{pmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{pmatrix} \quad \text{and} \quad \frac{1}{3}\begin{pmatrix} -\frac{1}{2} & 1 & -\frac{1}{2} \\ -2 & 4 & -2 \\ -\frac{1}{2} & 1 & -\frac{1}{2} \end{pmatrix}. \tag{1.3b}$$

When $\epsilon$ is small, the coupling in the $y$-direction is weak. This means that a standard point smoothing algorithm such as Gauss-Seidel will be ineffective at reducing errors which are smooth in the $x$-direction but oscillatory in the $y$-direction. This is not a problem if the multigrid coarse mesh is obtained by semi-coarsening, which coarsens only in the directions where the error after relaxation is smooth. Here, this implies coarsening only in the $x$-direction. To semi-coarsen, however, the strength of connection measure should determine that the coupling in the vertical direction is weak compared to the coupling in the horizontal direction. Unfortunately, the classic strength of connection measure only indicates modestly stronger coupling in the horizontal direction. Depending on the drop tolerance, the multigrid algorithm may or may not make the proper classification. The simple use of matrix coefficients is not sufficient to reliably reveal connection strength.

Another motivating concept for determining strength of connection has been the matrix inverse. The inverse can at first seem an attractive target for calculating strength of connection because the inverse relates the residual to the error,

$$A^{-1}r = e. \tag{1.4}$$

This relationship can appear useful for determining strength of connection in multigrid, because multigrid solves residual equations on coarse grids. However, the inverse does not necessarily give useful local strength of connection information. The information in the inverse is too global and includes information about both low and high energy modes.

For example, consider a standard 1-D finite difference approximation of

$$-\epsilon(x)u_{xx} = f \tag{1.5}$$

with 20 points on $[0, 1]$ and $h = 1/19$. Define a Neumann boundary condition at $x = 0.0$ and a Dirichlet boundary condition at $x = 1.0$. Let $\epsilon(x) = 0.001$ if $x \leq 0.5$ and $\epsilon(x) = 1.0$ otherwise. The 11th row of the matrix contains the stencil,

$$\begin{array}{rccc} & [-0.001 & 1.001 & -1] \\ x = & 10h & 11h & 12h \end{array}. \tag{1.6}$$

However, the stencil of the matrix inverse for the two nearest neighbors of point 11 is

$$\begin{array}{rccc} & [10.0 & 10.0 & 9.0] \\ x = & 10h & 11h & 12h \end{array}. \tag{1.7}$$

This is incorrect from a strength standpoint. The strong connection for point 11, should be to the right, in the direction of the large PDE coefficient. Instead, the strength information is inconclusive, and even hints at a slightly stronger connection in the direction of the small PDE coefficient. The reason for this can be explained by considering the Green's function. Since the finite difference stencil for the Neumann boundary condition sums to 0, the Green's function must be constant from point 11 to the Neumann condition on the left. On the other hand, the finite difference stencil for the Dirichlet boundary condition forces the Green's function to be zero at the Dirichlet boundary on the right. The Green's function corresponding to the 11th point is shown in Figure 1.1. It essentially corresponds to two linear functions,

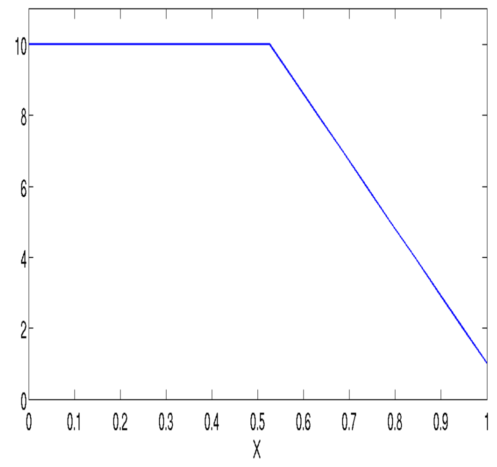


FIG. 1.1. *Column 11 of* $A^{-1}$

one to the left of point 11 and the other to the right of point 11, whose slopes are chosen to satisfy the boundary conditions, but contain no information about $\epsilon$.

If we continue examining this example, we can find even more problems with the inverse. Columns of the inverse corresponding to points to the left and to the right of the interface also give incorrect strength information. Even simpler examples can yield inverses with misleading strength information. Standard finite differencing applied to 1-D isotropic diffusion with both a Neumann and a Dirichlet boundary condition yields an inverse with misleading strength of connection information near the boundary conditions.

One recent idea that has attracted attention is Compatible Relaxation (CR), which is used to identify a subset of the original $n$ degrees of freedom which will define a coarse mesh. CR iteratively carries out a mock-AMG cycle on $Ae = 0$, where $e$ is an initial random guess. First, $e$ is smoothed with the multigrid smoother but each point in the tentative set of coarse points is made to be invariant and held to 0. The basic idea is that this models a perfect coarse level operator which reduces the error at the coarse points to zero. Second, CR augments the tentative set of coarse points with a maximal independent set of points that were not sufficiently reduced in $e$ by the mock-AMG cycle. The maximal independent set is chosen using the graph of the matrix. This process is repeated for $e$ until convergence is satisfactory. Multiple initial random guesses are tried and the algorithm stops once a good balance has been struck between the coarsening ratio and the convergence rate of the CR smoothing step. While CR does not explicitly make strength of connection decisions, it does make strength related decisions when choosing a coarse grid. However, choosing a coarse grid is essentially an easier problem than calculating strength.

Another strength of connection avenue that has been explored is based on local approximations to the matrix inverse [1, 2]. These methods follow the reasoning that strength of connections within the matrix inverse are the most relevant when determining intergrid transfers. While our examples illustrate that this is incorrect, local approximations to the inverse can be much better than the actual inverse. This is because the local approximation may not suffer from being too global. In this paper, the strength of connection measure in [1] is examined and referred to as the $\delta$-function inverse measure. This method uses relaxation to calculate approximations to the matrix inverse with a 0 initial guess. An energy-based post-processing step is then applied to each column of the approximate inverse to determine strength of connection.

Our central premise is that strength of connection information suitable for a multigrid algorithm is best determined by examining the evolution of an initial Dirac $\delta$-function during the standard multigrid relaxation process. Based on a relationship between weighted-Jacobi relaxation and the time marching of ordinary differential equations (ODEs), an ODE perspective is presented for understanding the $\delta$-function inverse measure and the evolution of delta functions during relaxation. The ODE perspective is used to shed new light on limitations associated with classical strength of connection measures as well as limitations associated with matrix inverses. In particular, classical strength of connection measures can be viewed as the initial evolution of a $\delta$-function within an ODE framework while matrix inverses are more closely tied to steady-state behavior. It is shown that the initial evolution of a $\delta$-function may be inaccurate due to high energy modes associated with discretization errors while the steady-state solution can be too global in nature to properly mimic the behavior of relaxation. A closely related modified measure to the one in [1] is then proposed based on the time evolution of a $\delta$-function until an intermediate time. A key issue in this proposed method is the determination of an appropriate intermediate time from which to base strength of connection.

In Section 2, an ODE perspective related to CR is presented that mirrors a Jacobi-relaxation type iteration. The ODE perspective provides an additional view on the limitations of classic strength measures and leads into the topic of matrix inverses and the $\delta$-function inverse measure. In Section 3, the $\delta$-function inverse measure is discussed from the ODE perspective. In Section 4, a new method related to the $\delta$-function inverse measure is proposed and analyzed from the ODE perspective. Experimental results are also then given along with a scale invariance proof.

**2. ODE Perspective On Strength.** An ordinary differential equation (ODE) perspective is presented. The perspective provides an additional view on the limitations of classical strength measures and leads into the topic of local approximations to matrix inverses and the delta-function inverse measure.

Consider the following ODE,

$$u_t = -Au, \text{ with } u(0) = \delta_i \tag{2.1}$$

where $\delta_i$ is a Dirac delta function centered at the location of the $i$-th grid point and $A$ is a symmetric positive definite matrix that is assumed to be diagonally scaled so that its diagonal entries are one. The exact solution to this system is

$$u = e^{-At}\delta_i. \tag{2.2}$$

When $A$ corresponds to a Laplacian, this ODE models the diffusion in time of an initial point distribution, $\delta_i$. Obviously, the steady state solution is just $u = 0$.

Numerically, equation (2.1) can be solved by the forward Euler method resulting in an iteration of the form

$$\begin{aligned} u^{(0)} &= \delta_i. \\ u^{(k+1)} &= u^{(k)} - \Delta t\, Au^{(k)}. \end{aligned} \tag{2.3a}$$

or

$$u^{(k)} = (I - \Delta t\, A)^k \delta_j. \tag{2.3b}$$

Strength of connection corresponds to how much a point $i$ influences a point $j$. In the context of (2.1), how does the $\delta$-function at point $i$ spread to point $j$. At $t = 0$, we have

$$u_t = -A\delta_i. \tag{2.4}$$

$A\delta_i$ is simply the matrix coefficients in the $i$-th row. That is, the growth of $u$ at $j$ for $t = 0$ is just given by the size of the coefficient $A(i, j)$. This is in principle identical to standard strength of connection measures. There is indeed a strong link between the matrix stencil and the time evolution of $\delta$-functions at $t = 0$.

As mentioned the steady state solution of equation (2.1) is just 0 and yields no useful information. Instead, suppose we consider the following ODE:

$$u_t = -Au + \delta_i, \text{ with } u(0) = 0. \tag{2.5}$$

When $A$ corresponds to a Laplacian, this ODE models diffusion in time with a constant source applied at $i$. This too might provide insight into how $i$ influences a point $j$ via the operator $A$. At $t = 0$, we have $u_t = \delta_i$. After one step of a forward Euler scheme, we have $u_t(\Delta t) = -\Delta t\ A\delta_i + \delta_i$. Thus, once again we have the influence of $i$ at the point $j$ governed primarily by the size of the matrix coefficients. The steady state solution is now given by $A^{-1}\delta_i$, i.e. a column of the inverse of A. However as already discussed, the matrix inverse can be misleading as it is too global.

Now, consider the transient solution of (1.2) with $\epsilon = 0$ given by,

$$u_t = -u_{xx} + \delta(x^*, y^*), \tag{2.6}$$

where $f$ is taken as a $\delta$-function centered at some spatial location given by $(x^*, y^*)$. The solution of the associated PDE (2.6) properly indicates that there is no coupling in the $y$-direction. The $\delta$-function only spreads in the $x$-direction as time increases. More precisely, $u(x, y) = 0$ for $x \neq x^*$ and $t > 0$. The associated PDE should give accurate strength of connection information at time $t = 0$. A problem can occur, however, when the spatial term is discretized. For example while discretizing the spatial term with Q1 basis functions on a uniform grid, one can intuitively see that the basis functions still interact in the weak direction and will hence yield some discretization error in the weak direction where there is no PDE coupling. This is reflected in the right-most expression of (1.3b) by the top three and bottom three stencil coefficients. We spare the reader the details, but the overall stencil yields $O(h^2)$ cross derivative truncation error terms, in the case of a sufficiently smooth function $u$. Normally, this error contribution is small for smooth functions. However, the $\delta$-functions are not smooth and so these error terms are significant during the initial time steps. Since there is no PDE coupling in the $y$-direction, any error terms involving derivatives taken in that direction are significant.

Thus in summary, solutions to equations (2.1) and (2.5) can be considered to generate strength of connection information. The solutions at $t = 0$ for (2.1) and at $t = \Delta t$ for equation (2.5) are similar to standard AMG strength of connection measures because they reduce to using the matrix stencil to make strength decisions. We know, however, that the use of the matrix stencil is quite limited due to discretization errors in directions of weak coupling within the PDE. Time evolution of the $\delta$-function can serve to damp the high frequency discretization errors and result in more accurate strength of connection information.

On the other hand, the $t \to \infty$ solutions give us either $A^{-1}$ or just 0. These solutions are often too global and do not accurately represent the local influence of $i$ on $j$. Instead, we will try and consider an intermediate time where local discretization errors in the weak direction have decayed sufficiently, but where the time is not too large to render the solution global. This will be discussed in Section 3.

**3. Matrix Inverses And The $\delta$-Function Inverse Measure.** As discussed, the matrix inverse is often too global to adequately capture strength of connection information. However, there has been some success in using approximate matrix inverses as a means of determining strength of connection in [1] and [2]. The success of these methods hinges on the fact that the matrix inverse is not actually computed, as this is too expensive. Instead, a local approximation to the matrix inverse is generated and this local approximation is actually better for determining strength of connection than the true matrix inverse. The local nature of the approximation indirectly accounts for the relaxation process used to compute it and ignores distorting boundary effects.

If, for example, we recall (2.5):

$$u_t = -Au + \delta_i, \text{ with } u(0) = 0$$

and solve this system by a forward Euler method. This essentially solves

$$Au = \delta_i \tag{3.1}$$

by a Jacobi scheme. This idea was considered in [1] along with the use of other iterative relaxation schemes.

The approximate column of the inverse was then combined with a particular strength formula involving $u$ and the $A$-norm. This energy-based post-processing step calculates strength of connection between $i$ and $j$ by taking the $i$-th approximate column of the inverse, $z_i$, and evaluating

$$\frac{||\bar{z}_i||_A - ||z_i||_A}{||z_i||_A}, \tag{3.2}$$

where $\bar{z}_i$ is $z_i$ zeroed out at entry $j$. This corresponds to calculating a normalized change in energy for $z_i$. Since $z_i$ is a locally smooth vector, it would make a good interpolation basis function around $i$. Hence if one zeroed out an entry, $j$, in $z_i$, i.e. interpolation from $j$ is not used, a large change in energy would be expected if $j$ were important to the interpolation. This indicates a strong connection.

In our experiments with the scheme, we found that it often worked well in practice but produced less accurate strength of connection measures if the iteration was carried on too long. The $\delta$-function inverse measure converges to $y(t) = A^{-1}\delta_i + e^{-At}\delta_i$. As the iterations count is increased for this method, $t \rightarrow \infty$ and all information about smooth modes in the exponential is lost. However, for small numbers of iterations, the result should be dominated by the locally smooth modes present in the matrix exponential.

The $\delta$-function inverse measure has a number of strengths and worked well in a number of our experiments. It worked as well as a distance-based strength measure on stretched meshes. It avoids any dependence on random starting guesses. Calculation (3.2) experimentally gave useful information, albeit at the cost of $n$ $A$-norms.

However, this method is also not without its weaknesses. Iterating $k$ times to generate the approximate column of the inverse is equivalent to $kn$ matrix-vector products. The method converges to non-ideal strength information, i.e. $A^{-1}$. The method does not indicate when it is appropriate to stop and if one iterates too much, the quality of the strength information degrades.

## 4. Proposed Method.

In this section, a new method is proposed by integrating the strengths of CR and the $\delta$-function inverse measure while addressing their weaknesses. We give a description for the method using the ODE perspective, show scale invariance and finally give experimental results.

### 4.1. Algorithm Description.

The proposed method solves equation (2.1) with Euler's method and $\delta_i$ as the initial guess. The steady state solution is 0, but the method stops at a "moderate" time, $t_f$. The strength information for $i$ is then in the resulting smoothed vector.

This vector, $(I - \Delta t D^{-1}A)^k \delta_i$, where $D = diag(A)$ and we have modified (2.3b) to explicitly account for scaling, can be examined directly to determine connection strength or it can be post-processed as in expression (3.2). The only parameters that could be user-defined are $k$ and $t_f$, i.e. the number of time steps and the final time. Here is a simple algorithm that describes our method.

```
Input:  $A$:          Matrix
        $t_f$:        Stop time for evolution of $\delta$-function
        $k$:          Time steps for evolution of $\delta$-function
        $drop\text{-}tol$: Drop tolerance for weak connections
        $energy$:     Boolean control for post-processing
        $b$:          Null Space vector that needs to be well approximated on
                      coarse meshes.  $b$ is often taken to be a vector of ones.
Output: $S$:          $S(i,j) = i$'s strength of connection to $j$
                      $S$ has the same sparsity structure as $A$

for $i$ = 1:$numRows$
    $z = (I - \frac{t_f}{k} D^{-1}A)^k \delta_i$
    $cols$ = nonzero-pattern($A(i,:)$)
    for $j$ = 1:length($cols$)
        if($energy$)
            %Energy-based Post-processing
            $\bar{z} = z$
            $\bar{z}(cols(j)) = 0$
            $S(i, cols(j)) = \frac{\|\bar{z}\|_A - \|z\|_A}{\|z\|_A}$
        else
            %No Post-processing
            $S(i, cols(j)) = \frac{z(cols(j))/b(cols(j))}{z(i)/b(i)}$
        end
        Apply($S(i,:)$, $drop\text{-}tol$)
```

The calculation $\frac{z(cols(j))/b(cols(j))}{z(i)/b(i)}$ serves two purposes. One, this calculation ensures scale invariance. Two, the comparison of $z$ to a null space vector, $b$, allows for this method to work for nonconstant null spaces. This calculation measures how well a locally smooth function around $i$ approximates the null space vector at neighbor $cols(j)$. This calculation should accurately determine strength of connection, i.e. how well algebraically smooth error can be interpolated from $i$ to $cols(j)$.

The choice of both $t_f$ and $k$ is not entirely clear. $k$ must at least be chosen so that the iteration is stable. We have, however, found that large $k$ rarely helps. $k = 1$ is similar to the classic strength measure in that only matrix coefficients are used. We do find that $k = 2$ offers significant improvement over $k = 1$ but that $k > 2$ does not offer much further improvement. With respect to $t_f$, a value too small will result in a measure close to the classic strength measure and is hence undesirable. A value too large will result in information that is too global in scope. Overall, we want a $t_f$ just large enough to damp discretization error. Also, $t_f$ and $k$ should result in an Euler's method whose action is commensurate with the smoother used in the multigrid cycle. Such a choice that meets these constraints and that worked well in our experiments, let $t_f = \frac{1}{\rho(D^{-1}A)}$ and $k = 2$.

**4.2. Scale Invariance.** As with the $\delta$-function inverse measure, our method is invariant with respect to an arbitrary symmetric diagonal scaling.

Proof. Let $\widetilde{A} = \hat{D}^{-1/2} A \hat{D}^{-1/2}$, for an arbitrary nonsingular diagonal matrix $\hat{D}$, $D = diag(A)$

and $\widetilde{D} = diag(\widetilde{A})$. Note that $\widetilde{D} = \hat{D}^{-1}D$. We first smooth $\delta_i$ an arbitrary number of times with $\widetilde{A}$ in order to derive a relationship to smoothing with $A$.

$$(I - \omega\widetilde{D}^{-1}\widetilde{A})^k\delta_i = (I - \omega\widetilde{D}^{-1}\widetilde{A})\ \hat{D}^{1/2}\hat{D}^{-1/2}\ (I - \omega\widetilde{D}^{-1}\widetilde{A})\ \hat{D}^{1/2}\hat{D}^{-1/2}\dots \tag{4.1a}$$
$$(I - \omega\widetilde{D}^{-1}\widetilde{A})\ \hat{D}^{1/2}\hat{D}^{-1/2}\ \delta_i$$
$$= (\hat{D}^{1/2} - \omega D^{-1}\hat{D}^{1/2}A)\ \hat{D}^{-1/2}\ (\hat{D}^{1/2} - \omega D^{-1}\hat{D}^{1/2}A)\ \hat{D}^{-1/2}\dots$$
$$(\hat{D}^{1/2} - \omega D^{-1}\hat{D}^{1/2}A)\ \hat{D}^{-1/2}\delta_i \tag{4.1b}$$
$$= \hat{D}^{1/2}(I - \omega D^{-1}A)(I - \omega D^{-1}A)\dots(I - \omega D^{-1}A)\hat{D}^{-1/2}\delta_i \tag{4.1c}$$
$$= \hat{D}^{-1/2}_{(i,i)}\ \hat{D}^{1/2}(I - \omega D^{-1}A)^k\delta_i, \tag{4.1d}$$

where $\hat{D}^{-1/2}_{(i,i)}$ is a scalar equal to the $i$-th diagonal entry. Let $z_i = (I - \omega D^{-1}A)^k\delta_i$, $\widetilde{z}_i = \hat{D}^{-1/2}_{(i,i)}\ \hat{D}^{1/2}(I - \omega D^{-1}A)^k\delta_i$, and $\widetilde{b} = \hat{D}^{1/2}b$ be the null space vector for $\widetilde{A}$.

We first consider the case with no post-processing.

$$S(i,j) = \frac{\frac{\widetilde{z}(j)}{\widetilde{b}(j)}}{\frac{\widetilde{z}(i)}{\widetilde{b}(i)}} = \frac{\frac{\hat{D}^{-1/2}_{(i,i)}\hat{D}^{1/2}_{(j,j)}z(j)}{\hat{D}^{1/2}_{(j,j)}b(j)}}{\frac{\hat{D}^{-1/2}_{(i,i)}\hat{D}^{1/2}_{(i,i)}z(i)}{\hat{D}^{1/2}_{(i,i)}b(i)}} = \frac{\frac{z(j)}{b(j)}}{\frac{z(i)}{b(i)}} \tag{4.1e}$$

Now, consider the case of energy-based post-processing. Let $\overline{\widetilde{z}} = \widetilde{z}$ but with the entry corresponding to neighbor $j$ zeroed out.

$$S(i,j) = \frac{||\overline{\widetilde{z}}_i||_{\widetilde{A}} - ||\widetilde{z}_i||_{\widetilde{A}}}{||\widetilde{z}_i||_{\widetilde{A}}} = \frac{||\overline{\widetilde{z}}_i||_{\widetilde{A}}}{||\widetilde{z}_i||_{\widetilde{A}}} - 1 \tag{4.1f}$$
$$= \frac{||\hat{D}^{-1/2}_{(i,i)}\ \hat{D}^{1/2}\overline{z}_i||_{\widetilde{A}}}{||\hat{D}^{-1/2}_{(i,i)}\ \hat{D}^{1/2}z_i||_{\widetilde{A}}} - 1 \tag{4.1g}$$
$$= \frac{|\hat{D}^{-1/2}_{(i,i)}|}{|\hat{D}^{-1/2}_{(i,i)}|}\ \frac{\overline{z}_i^T\hat{D}^{1/2}\hat{D}^{-1/2}A\hat{D}^{-1/2}\hat{D}^{1/2}\overline{z}_i}{z_i^T\hat{D}^{1/2}\hat{D}^{-1/2}A\hat{D}^{-1/2}\hat{D}^{1/2}z_i} - 1 \tag{4.1h}$$
$$= \frac{||\overline{z}_i||_A - ||z_i||_A}{||z_i||_A}\ \square \tag{4.1i}$$

**4.3. Experiments.** We implemented our algorithm as part of the existing ML smoothed aggregation framework and implemented the energy minimization algorithm of [3] for prolongator generation. In the below tables, "Energy-based Post-processing" and "No Post-processing" refer to options in the algorithm of Section 4.1 and "Steps" refers to the number of time steps used. $t_f = \frac{1}{\rho(D^{-1}A)}$, unless otherwise mentioned.

All of the below strength stencils are for the degree of freedom in the center of a $31 \times 31$ regular mesh. The data is presented as $3 \times 3$ arrays of values that represent the strength of connection values between the degree of freedom in question and its geometric neighbors above, below, to the left, to the right and diagonally offset. The degree of freedom in question is represented as "***", as no strength of connection information is needed between a point and itself.

First, we briefly examine the isotropic case with Q-1 elements on a uniform grid. As expected, the strength stencils are isotropic. In Tables 4.1 and 4.2, we present the "No Post-processing" and "Energy-based Post-processing" cases respectively.

TABLE 4.1
*Isotropic – Strength of Connection Stencils – No Post-processing*

| Steps = | 1 | | | 3 | | |
|---|---|---|---|---|---|---|
| | 0.0836 | 0.0836 | 0.0836 | 0.0547 | 0.0583 | 0.0547 |
| Stencils | 0.0836 | *** | 0.0836 | 0.0583 | *** | 0.0583 |
| | 0.0836 | 0.0836 | 0.0836 | 0.0547 | 0.0583 | 0.0547 |

TABLE 4.2
*Isotropic – Strength of Connection Stencils – Energy-based Post-processing*

| Steps = | 1 | | | 3 | | |
|---|---|---|---|---|---|---|
| | 0.0190 | 0.0381 | 0.0190 | 0.0141 | 0.0183 | 0.0141 |
| Stencils | 0.0381 | *** | 0.0381 | 0.0183 | *** | 0.0183 |
| | 0.0190 | 0.0381 | 0.0190 | 0.0141 | 0.0183 | 0.0141 |

Next, we again examine results for Q-1 elements on a uniform grid, but with anisotropy that is rotated by $\frac{\pi}{4}$ and vertically aligned anisotropy, corresponding to

$$-(c^2 + \epsilon s^2)u_{xx} - 2(1 - \epsilon)cs\ u_{xy} - (\epsilon c^2 + s^2)u_{yy} = f, \tag{4.2}$$

where $\epsilon = 0.001$, $c = \cos(\theta)$, $s = \sin(\theta)$ and $\theta$ is the angle of rotation.

In Table 4.3, we show the matrix stencils, which can be compared with the computed strength measures. In Tables 4.4–4.7, we present strength stencils for the "No Post- processing" and "Energy-based Post-processing" options for the vertically aligned anisotropy case and then the rotated anisotropy case. For the vertically aligned case, $t_f = \frac{2}{\rho(D^{-1}A)}$.

TABLE 4.3
*Original Matrix Stencils*

| Problem | Vertical Anisotropy | | | Anisotropy Rot. By $\frac{\pi}{4}$ | | |
|---|---|---|---|---|---|---|
| | -0.1668 | -0.6663 | -0.1668 | 0.0829 | -0.1668 | -0.4166 |
| Stencils | 0.3326 | 1.3346 | 0.3326 | -0.1668 | 1.3346 | -0.1668 |
| | -0.1668 | -0.6663 | -0.1668 | -0.4166 | -0.1668 | 0.0829 |

TABLE 4.4
*Vertical Anisotropy – Strength of Connection Stencils – No Post-processing*

| Steps = | 1 | | | 2 | | |
|---|---|---|---|---|---|---|
| | 0.0838 | 0.3345 | 0.0838 | 0.0278 | 0.2085 | 0.0278 |
| Stencils | -0.1670 | *** | -0.1670 | -0.0830 | *** | -0.0830 |
| | 0.0838 | 0.3345 | 0.0838 | 0.0278 | 0.2085 | 0.0278 |
| Steps = | 3 | | | 4 | | |
| | 0.0257 | 0.1951 | 0.0257 | 0.0245 | 0.1889 | 0.0245 |
| Stencils | -0.0772 | *** | -0.0772 | -0.0743 | *** | -0.0743 |
| | 0.0257 | 0.1951 | 0.0257 | 0.0245 | 0.1889 | 0.0245 |

It is important that the range for appropriate drop tolerances is large. Weak connections are defined to be less than the drop tolerance times the largest strength of connection value for that degree of freedom. For instance in Table 4.7, the strong connections are along the diagonal from the lower-left to the upper-right and these connections are a factor of 4 greater

TABLE 4.5
*Vertical Anisotropy – Strength of Connection Stencils – Energy-Based Post-processing*

| Steps = | | 1 | | | 2 | |
|---|---|---|---|---|---|---|
| | -0.0512 | -0.1082 | -0.0512 | -0.0065 | 0.2157 | -0.0065 |
| Stencils | -0.1057 | *** | -0.1057 | 0.0084 | *** | 0.0084 |
| | -0.0512 | -0.1082 | -0.0512 | -0.0065 | 0.2157 | -0.0065 |
| Steps = | | 3 | | | 4 | |
| | -0.0037 | 0.2061 | -0.0037 | -0.0026 | 0.2002 | -0.0026 |
| Stencils | 0.0146 | *** | 0.0146 | 0.0166 | *** | 0.0166 |
| | -0.0037 | 0.2061 | -0.0037 | -0.0026 | 0.2002 | -0.0026 |

TABLE 4.6
*Anisotropy Rotated by $\frac{\pi}{4}$ – Strength of Connection Stencils – No Post-processing*

| Steps = | | 1 | | | 2 | |
|---|---|---|---|---|---|---|
| | -0.0347 | 0.0698 | 0.1742 | -0.0226 | 0.0552 | 0.1280 |
| Stencils | 0.0698 | *** | 0.0698 | 0.0552 | *** | 0.0552 |
| | 0.1742 | 0.0698 | -0.0347 | 0.1280 | 0.0552 | -0.0226 |
| Steps = | | 3 | | | 4 | |
| | -0.0205 | 0.0520 | 0.1190 | -0.0196 | 0.0506 | 0.1151 |
| Stencils | 0.0520 | *** | 0.0520 | 0.0506 | *** | 0.0506 |
| | 0.1190 | 0.0520 | -0.0205 | 0.1151 | 0.0506 | -0.0196 |

TABLE 4.7
*Anisotropy Rotated by $\frac{\pi}{4}$ – Strength of Connection Stencils – Energy-Based Post-processing*

| Steps = | | 1 | | | 2 | |
|---|---|---|---|---|---|---|
| | -0.0019 | 0.0287 | 0.0861 | 0.0011 | 0.0181 | 0.0731 |
| Stencils | 0.0287 | *** | 0.0287 | 0.0181 | *** | 0.0181 |
| | 0.0861 | 0.0287 | -0.0019 | 0.0731 | 0.0181 | 0.0011 |
| Steps = | | 3 | | | 4 | |
| | 0.0012 | 0.0161 | 0.0669 | 0.0012 | 0.0152 | 0.0642 |
| Stencils | 0.0161 | *** | 0.0161 | 0.0152 | *** | 0.0152 |
| | 0.0669 | 0.0161 | 0.0012 | 0.0642 | 0.0152 | 0.0012 |

than the next strongest connection. Hence a drop tolerance greater than 0.25 would be appropriate. In Table 4.4, the strong connections are in the vertical direction and are a factor of 7-8 greater than the next strongest connections, which imply a drop tolerance greater than $\frac{1}{7}$ is appropriate.

It is not entirely clear how to interpret the negative entries, but they most likely imply a weak connection. Also, it is important that the separation between weak and strong connections does not change much after 2 time steps. We therefore suggest using only 2 time steps with this method.

As a means of comparison, strength of connection information from the $\delta$-function inverse measure is given in Tables 4.8 and 4.9. One of the inherent problems of the $\delta$-function inverse measure appears in Table 4.9, where the strength of connection information begins to degrade for higher degrees. This also happens in the vertically aligned anisotropy case, but at higher degrees. It is important that this phenomenon was not observed in our method. The separation between strong and weak connections in Tables 4.8 and 4.9 and is roughly the same as in our method.

TABLE 4.8
*Vertical Anisotropy – Strength of Connection Stencils – $\delta$-function inverse measure*

| Steps = | 1 | | | 2 | | |
|---|---|---|---|---|---|---|
| | 0.0020 | 0.0541 | 0.0020 | 0.0017 | 0.0977 | 0.0017 |
| Stencils | 0.0113 | *** | 0.0113 | 0.0161 | *** | 0.0161 |
| | 0.0020 | 0.0541 | 0.0020 | 0.0017 | 0.0977 | 0.0017 |
| Steps = | 3 | | | 4 | | |
| | 0.0011 | 0.1359 | 0.0011 | 0.0006 | 0.1710 | 0.0006 |
| Stencils | 0.0187 | *** | 0.0187 | 0.0205 | *** | 0.0205 |
| | 0.0011 | 0.1359 | 0.0011 | 0.0006 | 0.1710 | 0.0006 |

TABLE 4.9
*Anisotropy Rotated by $\frac{\pi}{4}$ – Strength of Connection Stencils – $\delta$-function inverse measure*

| Steps = | 1 | | | 2 | | |
|---|---|---|---|---|---|---|
| | 0.0011 | 0.0073 | 0.0378 | 0.0012 | 0.0151 | 0.0634 |
| Stencils | 0.0073 | *** | 0.0073 | 0.0151 | *** | 0.0151 |
| | 0.0378 | 0.0073 | 0.0011 | 0.0634 | 0.0151 | 0.0012 |
| Steps = | 3 | | | 4 | | |
| | 0.0010 | 0.0230 | 0.0840 | 0.0008 | 0.0308 | 0.1021 |
| Stencils | 0.0230 | *** | 0.0230 | 0.0308 | *** | 0.0308 |
| | 0.0840 | 0.0230 | 0.0010 | 0.1021 | 0.0308 | 0.0008 |

If appropriate drop tolerance values are chosen, the above strength stencils will yield correct coarse grids. With correct coarse grids, AMG can be used as an effective preconditioner as evidenced in Table 4.10, where an AMG method was used in conjunction with the strength of connection measures computed here.

TABLE 4.10
*PCG Convergence Counts*

| Problem | | Vertical Ani. | Rot. By $\frac{\pi}{4}$ Ani. | Rot. By $\frac{\pi}{8}$ Ani. |
|---|---|---|---|---|
| | $31 \times 31$ | 12 | 9 | 12 |
| Mesh Size | $63 \times 63$ | 16 | 10 | 16 |
| | $127 \times 127$ | 17 | 14 | 20 |

**5. Conclusions.** The proposed method performs as well as the $\delta$-function inverse measure if the same number of iterations and energy-based postprocessing are both used. The proposed method can also be computationally much cheaper than the $\delta$-function inverse measure, especially if no energy-based post processing is used and the number of time steps is 2. In this case, the proposed method only calculates the entries of $(I - \Delta t\ A)^2\delta_i$ for the neighbors of $i$ in the matrix graph. The entire vector need not be calculated. However in the $\delta$-function inverse measure, the entire vector must be calculated so that the energy-based post-processing step can be applied. While the strength information produced by only 2 time steps is inferior to that produced by the $\delta$-function inverse measure using 2 time steps and energy-based post-processing, it is much more computationally feasible and is an improvement over just the matrix stencil.

As the iterations of the $\delta$-function inverse measure increase, $t_f \to \infty$, and the method converges to the matrix inverse, which is problematic. On the other hand, as the iterations of the proposed method increase, $\Delta t \to 0$, and the method converges to $e^{-At_f}\delta_i$, which is a useful combination of locally smooth modes for "moderate" $t_f$.

The proposed method is similar to CR in that we relax an initial error for the homogeneous system of equations and our strength decisions are based directly on the action of the smoother. However, we avoid the use of an undetermined number of random starting vectors by choosing point-sources as our initial error.

## REFERENCES

[1] J. Brannick, M. Brezina, S. MacLachlan, T. Manteuffel, and S. McCormick, *An energy-based AMG coarsening strategy*, Numer. Linear Algebra Appl., 13 (2006), pp. 133–148.

[2] O. Bröker, *Parallel Multigrid Methods Using Sparse Approximate Inverses*, PhD thesis, Dept. of Computer Science, ETH Zürich, May 2003.

[3] J. Mandel, M. Brezina, and P. Vanek, *Energy optimization of algebraic multigrid bases*, Computing, 62 (1999), pp. 205–228.

[4] J. W. Ruge and K. Stüben, *Algebraic multigrid (AMG)*, in Multigrid Methods, S. F. McCormick, ed., Frontiers Appl. Math., SIAM, Philadelphia, 1987, pp. 73–130.

[5] P. Vaněk, J. Mandel, and M. Brezina, *Algebraic multigrid based on smoothed aggregation for second and fourth order problems*, Computing, 56 (1996), pp. 179–196.

# PRELIMINARY INFRASTRUCTURES FOR INTEGRATING MODEL ORDER REDUCTION METHODS INTO XYCE™

RYAN NONG[*] AND HEIDI THORNQUIST[†]

**Abstract.** While advances in manufacturing enable the fabrication of integrated circuits containing tens-to-hundreds of millions of devices, the time-sensitive modeling and simulation necessary to design these circuits poses a significant computational challenge. Model-order-reduction techniques attempt to alleviate the computational difficulties by generating macromodels that capture the desired input-output behavior of larger dynamical systems. Even though model order reduction is an active area of research in design automation, the techniques see limited use in commercial circuit design tools and mostly for interconnect macromodeling. The Xyce circuit simulator is focused on developing the capability to solve extremely large circuit problems that have tens-to-hundreds of millions of devices, which motivates the research and integration of broadly-applicable model-order-reduction techniques. This paper lays out the groundwork for the integration of two recent model-order-reduction techniques into Xyce.

**1. Introduction.** Advances in manufacturing enable the fabrication of integrated circuits containing tens-to-hundreds of millions of devices. However, the time-sensitive modeling and simulation necessary to design these circuits poses a significant computational challenge. When the integrated circuit has millions of devices, performing a full system simulation is practically infeasible. The principal reason for this is the time required for the nonlinear solver to compute the solutions of large linearized systems during the simulation of these circuits.

Model-order-reduction (MOR) techniques attempt to produce macromodels that capture the desired characteristics, such as passivity or stability, of larger dynamical systems while enabling substantial speedups in simulation time. The vast majority of current MOR techniques are projection based, meaning that a macromodel of the large-scale dynamical system is generated by projecting it onto some low-dimensional subspace. Projection based MOR methods generate their subspace using either a moment matching based method (Krylov subspace method) or SVD based method (balanced realization, proper orthogonal decomposition). While model-order reduction is an active area of research, the techniques see limited use in commercial Electrical Design Automation (EDA) tools and are mostly used for interconnect macromodeling.

In this paper, we present a preliminary study of the infrastructure required for the integration of current MOR methods into Xyce, a modern circuit simulation code. We will discuss how Xyce and other modern circuit simulators formulate the circuit equations and the structure of the resulting set of differential algebraic equations (DAEs) in Section 2. Given this knowledge, we then assess the direct applicability of current MOR methods to modern circuit simulation codes. To make this assessment more manageable we consider two passivity preserving MOR methods: the Structure-Preserving Reduced-Order Interconnect Macromodeling (SPRIM) method by Freund [2], presented in Section 3.1, and the invariant subspace method by Sorensen [7] and Nong [5], presented in Section 3.2. Implementing these methods using C++ and Trilinos [3] is discussed in Section 4, where any modifications to either the circuit simulator or MOR method that would be required for integration are also presented. Numerical results from the application of both methods to an RLC ladder circuit are given in Section 5. This preliminary study has provided insight into the interaction of modern circuit simulators and MOR methods, but has illuminated a host of issues that need to be addressed by both, which are briefly discussed in Section 6.

In this paper, except when specified otherwise, upper case bold letters (**A**, **B**, etc.) will

[*]CAAM Department, Rice University, ryannong@caam.rice.edu
[†]Sandia National Laboratories, hkthorn@sandia.gov

denote matrices, lower case bold letters ($\mathbf{x}$, $\mathbf{y}$, etc.) will denote vectors, and non-bold or Greek letters will denote scalars. Script letters ($\mathcal{A}$, $\mathcal{E}$, etc.) will denote special or structured matrices. Conjugate transpose is denoted by $\mathbf{A}^*$ and transpose by $\mathbf{A}^T$.

## 2. Xyce.

**2. Xyce.** Xyce is a massively parallel SPICE-style circuit simulator developed to support the needs of electrical designers at Sandia National Laboratories. To this end, Xyce development is focused on improving capability over the current state-of-the-art in several areas of circuit simulation. One such area is the capability to solve extremely large circuit problems that have tens-to-hundreds of millions of devices. This certainly motivates an efficient parallel implementation that can take advantage of the powerful computing resources at Sandia National Laboratories. However, this also motivates the research on performance improvements for important numerical kernels, often requiring state-of-the-art algorithms and novel solution techniques like model-order reduction.

Xyce and many other modern circuit simulators use modified nodal analysis (MNA) to solve their circuit problems (see for instance Keiter et al. [4]). This formulation is based on the three types of equations found in circuit theory:

- **Kirchhoff's voltage law (KVL)**, which specifies that the sum of the branch voltage drops around a closed loop of a circuit should equal zero. This is expressed as $\sum_{j=0}^{B_1} v_j = 0$, where $B_1$ is the number of branches in a closed loop.
- **Kirchhoff's current law (KCL)**, which specifies that at any node in a circuit the sum of the branch currents into/out of the node must equal zero. This is expressed as $\sum_{j=0}^{N_1} i_j = 0$, where $N_1$ is the number of branch currents into/out of a circuit node.
- **Branch constitutive equations/relationships (BCEs/BCRs)**, which describe the physical behavior of the circuit elements. For resistors, capacitors, and inductors the BCEs/BCRs are, respectively,

$$i = Gv = \frac{v}{R}, \quad i = C\frac{dv}{dt}, \quad v = L\frac{di}{dt}.$$

MNA is sometimes referred to as the "modified KCL formulation" because it satisfies one KCL equation at every node, except the ground node, and adds on at least one equation that is not a KCL equation. These equations correspond to non-Ohmic devices, like an independent voltage source or inductor, and the variables added to satisfy these equations are generally current variables. Once MNA has been performed on a circuit, the result is a state space model in descriptor form

$$\begin{aligned} \mathbf{E}\tfrac{d\mathbf{x}}{dt} &= \mathbf{A}\mathbf{x}(t) + \mathbf{B}\mathbf{u}(t) \\ \mathbf{y}(t) &= \mathbf{C}\mathbf{x}(t) + \mathbf{D}\mathbf{u}(t) \end{aligned} \tag{2.1}$$

where $\mathbf{E}$, $\mathbf{A} \in \mathbb{R}^{n\times n}$, $\mathbf{B} \in \mathbb{R}^{n\times p}$, $\mathbf{C} \in \mathbb{R}^{p\times n}$, $\mathbf{D} \in \mathbb{R}^{p\times p}$, $\mathbf{x}(t) \in \mathbb{R}^n$ is the state, $\mathbf{u}(t) \in \mathbb{R}^p$ is the input, and $\mathbf{y}(t) \in \mathbb{R}^p$ is the output. The order of the system is $n$ and number of inputs (outputs) is $p$.

**3. Methods.** In this section we present the two passivity preserving algorithms for model-order reduction that are under consideration for integration into Xyce. First we discuss the SPRIM method which was introduced by Freund [2] in Section 3.1. Then we discuss the invariant subspace method which was proposed by Sorensen [7] and further studied by Nong [5] in Section 3.2.

**3.1. SPRIM.** The SPRIM algorithm was proposed by Freund [2] in 2004 as an improvement of the Passive Reduced-Order Interconnect Macromodeling Algorithm (PRIMA) by Odabasioglu et al. [6]. In addition to preserving passivity for linear time-invariant (LTI) systems as assured in the PRIMA, the SPRIM algorithm preserves other structures such as reciprocity or the block structure of the circuit matrices inherent to RLC circuits. Moreover, with respect to the original models, the reduced models resulting from the SPRIM algorithm match twice as many moments as those resulting from the PRIMA with the same computational work. Since it is based on the PRIMA, the SPRIM algorithm also is concerned with only symmetric positive definite (SPD) systems obtained from a time domain MNA formulation.

We will now briefly present the formulation of the RLC circuit equations that is required by the SPRIM algorithm; more detail can be found in [2]. First, let $\mathbf{E}$ be the adjacency matrix of the directional graph which describes the connectivity of an RLC circuit. The rows and columns of $\mathbf{E}$ correspond to the graph edges (circuit elements) and graph nodes (circuit nodes), respectively. By convention, a row of $\mathbf{E}$ contains $+1$ in the column corresponding to the source node, $-1$ in the column corresponding to the destination node, and 0 everywhere else. In addition, the column corresponding to the ground node of the circuit is omitted in order to remove redundancy.

Denote by $\mathbf{v}_n$ and $\mathbf{i}_b$ the node voltages and branch currents of the circuit. The subscript $b$ can be further categorized into $i$, $g$, $c$ and $l$ which represent branches containing current sources, resistors (conductors), capacitors and inductors, respectively. (Note that in the following derivation, only current sources are considered.) Using Kirchhoff's laws, we arrive at the following:

$$\begin{aligned} \text{KCL:} \quad & \mathbf{E}^T\mathbf{i}_b = 0 \\ \text{KVL:} \quad & \mathbf{E}\mathbf{v}_n = \mathbf{v}_b. \end{aligned} \tag{3.1}$$

In addition to (3.1), partitioning $\mathbf{E}$, $\mathbf{v}_b$ and $\mathbf{i}_b$ as follows,

$$\mathbf{E} = \begin{bmatrix} \mathbf{E}_i \\ \mathbf{E}_g \\ \mathbf{E}_c \\ \mathbf{E}_l \end{bmatrix}, \quad \mathbf{v}_b = \begin{bmatrix} \mathbf{v}_i \\ \mathbf{v}_g \\ \mathbf{v}_c \\ \mathbf{v}_l \end{bmatrix}, \quad \mathbf{i}_b = \begin{bmatrix} \mathbf{i}_i \\ \mathbf{i}_g \\ \mathbf{i}_c \\ \mathbf{i}_l \end{bmatrix}$$

and considering the corresponding BCRs

$$\mathbf{i}_i = -\mathbf{I}_t(t), \quad \mathbf{i}_g = \mathbf{G}\mathbf{v}_g, \quad \mathbf{i}_c = \mathbf{C}\frac{d}{dt}\mathbf{v}_c, \quad \mathbf{v}_l = \mathbf{L}\frac{d}{dt}\mathbf{i}_l \tag{3.2}$$

give the following MNA formulation of the circuit equations:

$$\mathcal{G}\mathbf{x} + \mathcal{C}\frac{d}{dt}\mathbf{x} = \mathcal{B}\mathbf{I}_t(t), \tag{3.3}$$

where $\mathcal{G} = \begin{bmatrix} \mathbf{E}_g^T\mathbf{G}\mathbf{E}_g & \mathbf{E}_l^T \\ -\mathbf{E}_l & \mathbf{0} \end{bmatrix}$, $\mathbf{x} = \begin{bmatrix} \mathbf{v}_n \\ \mathbf{i}_l \end{bmatrix}$, $\mathcal{C} = \begin{bmatrix} \mathbf{E}_c^T\mathbf{C}\mathbf{E}_c & \mathbf{0} \\ \mathbf{0} & \mathbf{L} \end{bmatrix}$, $\mathcal{B} = \begin{bmatrix} \mathbf{E}_i^T \\ \mathbf{0} \end{bmatrix}$ and $\mathbf{I}_t(t)$ is the vector of current source values, $\mathbf{G}$, $\mathbf{C}$ SPD matrices and $\mathbf{L}$ a symmetric positive semidefinite matrix whose entries are the conductance, capacitance and inductance values of the elements, respectively.

Laplace transforming (3.3) and assuming zero initial conditions give

$$\begin{aligned} (\mathcal{G} + s\mathcal{C})\mathbf{X} &= \mathcal{B}\mathbf{I}_s(s) \\ \mathbf{V}_i &= \mathbf{B}^T\mathbf{X} \end{aligned} \tag{3.4}$$

where $\mathbf{X}$, $\mathbf{I}_s(s)$ and $\mathbf{V}_i$ are the Laplace transforms of the state vector $\mathbf{x}$, the vector of current source values $\mathbf{I}_t(t)$ and the vector of voltages across the excitation sources, respectively. Then the transfer function of the circuit is as follows,

$$\mathbf{G}(s) = \mathcal{B}^T(\mathcal{G} + s\mathcal{C})^{-1}\mathcal{B}. \tag{3.5}$$

The goal is to construct a projection matrix $\mathbf{V}_n$ whose columns span a Krylov subspace of dimension $n$ so that the reduced model of order $n$ can be formed as the projection of the original model on this Krylov subspace. The SPRIM algorithm is as follows,

**Algorithm 1** *SPRIM Algorithm*

1. *Obtain an expansion point $s_0$ and the following block matrices from a MNA formulation*

$$\mathcal{G} = \begin{bmatrix} \mathbf{G}_1 & \mathbf{G}_2^T \\ -\mathbf{G}_2 & \mathbf{0} \end{bmatrix}, \quad \mathcal{C} = \begin{bmatrix} \mathbf{C}_1 & \mathbf{0} \\ \mathbf{0} & \mathbf{C}_2 \end{bmatrix}, \quad \mathcal{B} = \begin{bmatrix} \mathbf{B}_1 \\ \mathbf{0} \end{bmatrix},$$

*where $\mathbf{G}_1 \geq \mathbf{0}$, $\mathbf{C}_1 \geq \mathbf{0}$ and $\mathbf{C}_2 \succ \mathbf{0}$.*

2. *Formally set $\mathcal{A} = -(\mathcal{G} + s_0\mathcal{C})^{-1}\mathcal{C}$ and $\mathcal{R} = (\mathcal{G} + s_0\mathcal{C})^{-1}\mathcal{B}$.*
3. *Use a block Krylov subspace method to construct a projection matrix $\mathbf{V}_n$ such that $\mathbf{V}_n = [\mathbf{v}_1\ \mathbf{v}_2\ \ldots\ \mathbf{v}_n]$, where $span(\mathbf{V}_n) = \mathcal{K}_n(\mathcal{A}, \mathcal{R})$.*
4. *Partition $\mathbf{V}_n$ in accordance with the block structure of $\mathcal{G}$ as follows*

$$\mathbf{V}_n = \begin{bmatrix} \mathbf{V}_1 \\ \mathbf{V}_2 \end{bmatrix}$$

5. *Compute $\tilde{\mathbf{G}}_1 = \mathbf{V}_1^T\mathbf{G}_1\mathbf{V}_1$, $\tilde{\mathbf{G}}_2 = \mathbf{V}_2^T\mathbf{G}_2\mathbf{V}_1$, $\tilde{\mathbf{C}}_1 = \mathbf{V}_1^T\mathbf{C}_1\mathbf{V}_1$, $\tilde{\mathbf{C}}_2 = \mathbf{V}_2^T\mathbf{C}_2\mathbf{V}_2$, $\tilde{\mathbf{B}}_1 = \mathbf{V}_1^T\mathbf{B}_1$ and form*

$$\tilde{\mathcal{G}}_n = \begin{bmatrix} \tilde{\mathbf{G}}_1 & \tilde{\mathbf{G}}_2^T \\ -\tilde{\mathbf{G}}_2 & \mathbf{0} \end{bmatrix}, \quad \tilde{\mathcal{C}}_n = \begin{bmatrix} \tilde{\mathbf{C}}_1 & \mathbf{0} \\ \mathbf{0} & \tilde{\mathbf{C}}_2 \end{bmatrix}, \quad \tilde{\mathcal{B}}_n = \begin{bmatrix} \tilde{\mathbf{B}}_1 \\ \mathbf{0} \end{bmatrix}.$$

6. *The transfer function of the reduced-order model of order n is as follows,*

$$\tilde{G}_n(s) = \tilde{\mathcal{B}}_n^T(\tilde{\mathcal{G}}_n + s\tilde{\mathcal{C}}_n)^{-1}\tilde{\mathcal{B}}_n. \tag{3.6}$$

**3.2. Invariant Subspace.** The invariant subspace algorithm was proposed by Sorensen [7] in 2005 and further studied by Nong [5] in 2007. In contrast to the PRIMA and SPRIM methods, this algorithm does not restrict its application to SPD models. The algorithm assumes a state space realization $(\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D})$ of an LTI circuit resulting in the following linear dynamical system of equations:

$$\begin{aligned} \dot{\mathbf{x}} &= \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u} \\ \mathbf{y} &= \mathbf{C}\mathbf{x} + \mathbf{D}\mathbf{u}, \end{aligned} \tag{3.7}$$

where $\mathbf{A} \in \mathbb{R}^{n\times n}$, $\mathbf{B} \in \mathbb{R}^{n\times p}$, $\mathbf{C} \in \mathbb{R}^{p\times n}$ and $\mathbf{D} \in \mathbb{R}^{p\times p}$ and $\mathbf{x}(t)$ is the state, $\mathbf{u}(t)$ the input and $\mathbf{y}(t)$ the output of the system. In addition, $n$ is the order of the system and $p$ the number of inputs (outputs). The invariant subspace algorithm assures passivity preservation. At the current stage of the algorithm, only non-descriptor systems are considered.

The following is a brief description of the invariant subspace algorithm [7], which will lead into a discussion of subspace selection criteria and the two-stage reduction algorithm [5]. Consider an LTI system $\Sigma$ of order $n$ whose state space realization is $(\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D})$ as specified in (3.7) . The transfer function of $\Sigma$ is

$$\mathbf{G}(s) = \mathbf{C}(s\mathbf{I} - \mathbf{A})^{-1}\mathbf{B} + \mathbf{D}. \tag{3.8}$$

The goal is to construct a pair of projection matrices $\mathbf{V}$ and $\mathbf{W}$ whose columns span two $k$-dimensional subspaces $\mathcal{K}$ and $\mathcal{L}$ of $\mathbb{R}^n$ such that the reduced model $\hat{\Sigma}$ of order $k$ can be formed as the projection of the original $\Sigma$ on $\mathcal{K}$ and the resulting residual is orthogonal to $\mathcal{L}$. The invariant subspace method by Sorensen [7] transforms the model-order reduction problem into a highly-structured generalized eigenvalue problem and is as follows,

**Algorithm 2** *Invariant Subspace Algorithm*

1. *Formally construct the generalized eigenvalue problem* $(\mathcal{A}, \mathcal{E})$ *using the state space realization* $(\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D})$ *of* $\Sigma$*, where* $\mathbf{D} \succ \mathbf{0}$ *and*

$$\mathcal{A} = \begin{bmatrix} \mathbf{A} & & \mathbf{B} \\ & -\mathbf{A}^T & -\mathbf{C}^T \\ \mathbf{C} & \mathbf{B}^T & \mathbf{D}+\mathbf{D}^T \end{bmatrix}, \quad \mathcal{E} = \begin{bmatrix} \mathbf{I} & & \\ & \mathbf{I} & \\ & & \mathbf{0} \end{bmatrix}.$$

2. *Compute a* $k^{th}$*-order partial real Schur decomposition* $\mathcal{A}\mathbf{Q} = \mathcal{E}\mathbf{Q}\mathbf{R}$.
3. *Partition* $\mathbf{Q}$ *in accordance with the block structure of* $\mathcal{A}$ *as follows,*

$$\mathbf{Q} = \begin{bmatrix} \mathbf{X} \\ \mathbf{Y} \\ \mathbf{Z} \end{bmatrix}.$$

4. *Compute the singular value decomposition of* $\mathbf{X}^T\mathbf{Y}$ *as follows,*

$$\mathbf{X}^T\mathbf{Y} = \mathbf{Q}_x\mathbf{S}^2\mathbf{Q}_y^T.$$

5. *Compute* $\mathbf{V} = \mathbf{X}\mathbf{Q}_x\mathbf{S}^{-1}$*,* $\mathbf{W} = \mathbf{Y}\mathbf{Q}_y\mathbf{S}^{-1}$ *and form*

$$\hat{\mathbf{A}} = \mathbf{W}^T\mathbf{A}\mathbf{V}, \quad \hat{\mathbf{B}} = \mathbf{W}^T\mathbf{B}, \quad \hat{\mathbf{C}} = \mathbf{C}\mathbf{V}.$$

6. *The transfer function of the reduced-order model* $\hat{\Sigma}$ *of order k is as follows,*

$$\hat{\mathbf{G}}(s) = \hat{\mathbf{C}}(s\mathbf{I} - \hat{\mathbf{A}})^{-1}\hat{\mathbf{B}} + \mathbf{D}. \tag{3.9}$$

As mentioned above, the reduced-order model $\hat{\Sigma}$ is assured to be passive. However, different choices of $\mathbf{V}$ and $\mathbf{W}$ result in different reduced-order models some of which are not good approximations to the original system at all. Note that $\mathbf{V}$ and $\mathbf{W}$ result from $\mathbf{Q}$, the orthonormal matrix in the $k^{th}$-order partial real Schur decomposition. Thus, different invariant subspaces corresponding to different selections of $k$ out of $n$ finite stable generalized eigenvalues of $(\mathcal{A}, \mathcal{E})$ result in different reduced-order models.

In order to assure the reduced-order models to be good approximations to the originals, selection criteria for the finite stable generalized eigenvalues of $(\mathcal{A}, \mathcal{E})$ therefore must be developed. Moreover, for the case of large-scale systems where the invariant subspaces should be computed using iterative methods, not all selection criteria can be satisfied. To meet these requirements, Nong [5] proposes an algorithm called the two-stage reduction algorithm which works in large-scale setting and results in reduced-order models as good approximations to the originals. Given $\Sigma$, an LTI system of order $n$, the algorithm is as follows,

**Algorithm 3** *Two-Stage Reduction Algorithm*

1. *Formally construct the generalized eigenvalue problem* $(\mathcal{A}_1, \mathcal{E}_1)$ *using the state space realization* $(\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D})$ *of* $\Sigma$*, where* $\mathbf{D} \succ \mathbf{0}$ *and*

$$\mathcal{A}_1 = \begin{bmatrix} \mathbf{A} & & \mathbf{B} \\ & -\mathbf{A}^T & -\mathbf{C}^T \\ \mathbf{C} & \mathbf{B}^T & \mathbf{D}+\mathbf{D}^T \end{bmatrix}, \quad \mathcal{E}_1 = \begin{bmatrix} \mathbf{I} & & \\ & \mathbf{I} & \\ & & \mathbf{0} \end{bmatrix}.$$

2. *Compute an* $m^{th}$*-order partial real Schur decomposition* $\mathcal{A}_1\mathbf{Q}_1 = \mathcal{E}_1\mathbf{Q}_1\mathbf{R}_1$ *corresponding to the subset of m finite stable generalized eigenvalues with smallest modulus of* $(\mathcal{A}_1, \mathcal{E}_1)$.
3. *Partition* $\mathbf{Q}_1$ *in accordance with the block structure of* $(\mathcal{A}_1, \mathcal{E}_1)$ *as follows,*

$$\mathbf{Q}_1 = \begin{bmatrix} \mathbf{X}_1 \\ \mathbf{Y}_1 \\ \mathbf{Z}_1 \end{bmatrix}.$$

4. *Compute the singular value decomposition of* $\mathbf{X}_1^T\mathbf{Y}_1$ *as follows,*

$$\mathbf{X}_1^T\mathbf{Y}_1 = \mathbf{Q}_{x1}\mathbf{S}_1^2\mathbf{Q}_{y1}^T.$$

5. *Compute* $\mathbf{V}_1 = \mathbf{X}_1\mathbf{Q}_{x1}\mathbf{S}_1^{-1}$, $\mathbf{W}_1 = \mathbf{Y}_1\mathbf{Q}_{y1}\mathbf{S}_1^{-1}$ *and form*

$$\tilde{\mathbf{A}} = \mathbf{W}_1^T\mathbf{A}\mathbf{V}_1, \quad \tilde{\mathbf{B}} = \mathbf{W}_1^T\mathbf{B}, \quad \tilde{\mathbf{C}} = \mathbf{C}\mathbf{V}_1$$

*as part of the state space realization* $(\tilde{\mathbf{A}}, \tilde{\mathbf{B}}, \tilde{\mathbf{C}}, \mathbf{D})$ *of the intermediate reduced-order model* $\tilde{\Sigma}$ *of order m.*
6. *Formally construct the generalized eigenvalue problem* $(\mathcal{A}_2, \mathcal{E}_2)$ *using the state space realization* $(\tilde{\mathbf{A}}, \tilde{\mathbf{B}}, \tilde{\mathbf{C}}, \mathbf{D})$ *of* $\tilde{\Sigma}$, *where* $\mathbf{D} \succ \mathbf{0}$ *and*

$$\mathcal{A}_2 = \begin{bmatrix} \tilde{\mathbf{A}} & & \tilde{\mathbf{B}} \\ & -\tilde{\mathbf{A}}^T & -\tilde{\mathbf{C}}^T \\ \tilde{\mathbf{C}} & \tilde{\mathbf{B}}^T & \mathbf{D}+\mathbf{D}^T \end{bmatrix}, \quad \mathcal{E}_2 = \begin{bmatrix} \mathbf{I} & & \\ & \mathbf{I} & \\ & & \mathbf{0} \end{bmatrix}.$$

7. *Compute all the finite stable generalized eigenvalues of* $(\mathcal{A}_2, \mathcal{E}_2)$ *and their residues.*
8. *Compute an* $k^{th}$*-order partial real Schur decomposition* $\mathcal{A}_2\mathbf{Q}_2 = \mathcal{E}_2\mathbf{Q}_2\mathbf{R}_2$ *corresponding to the subset of k finite stable generalized eigenvalues with largest residue of* $(\mathcal{A}_2, \mathcal{E}_2)$.
9. *Partition* $\mathbf{Q}_2$ *in accordance with the block structure of* $\mathcal{A}_2$ *as follows,*

$$\mathbf{Q}_2 = \begin{bmatrix} \mathbf{X}_2 \\ \mathbf{Y}_2 \\ \mathbf{Z}_2 \end{bmatrix}.$$

10. *Compute the singular value decomposition of* $\mathbf{X}_2^T\mathbf{Y}_2$ *as follows,*

$$\mathbf{X}_2^T\mathbf{Y}_2 = \mathbf{Q}_{x2}\mathbf{S}_2^2\mathbf{Q}_{y2}^T.$$

11. *Compute* $\mathbf{V}_2 = \mathbf{X}_2\mathbf{Q}_{x2}\mathbf{S}_2^{-1}$, $\mathbf{W}_2 = \mathbf{Y}_2\mathbf{Q}_{y2}\mathbf{S}_2^{-1}$ *and form*

$$\hat{\mathbf{A}} = \mathbf{W}_2^T\tilde{\mathbf{A}}\mathbf{V}_2, \quad \hat{\mathbf{B}} = \mathbf{W}_2^T\tilde{\mathbf{B}}, \quad \hat{\mathbf{C}} = \tilde{\mathbf{C}}\mathbf{V}_2$$

*as part of the state space realization* $(\hat{\mathbf{A}}, \hat{\mathbf{B}}, \hat{\mathbf{C}}, \mathbf{D})$ *of the final reduced-order model* $\hat{\Sigma}$ *of order k.*
12. *The transfer function of the final reduced-order model* $\hat{\Sigma}$ *of order k is as follows,*

$$\hat{\mathbf{G}}(s) = \hat{\mathbf{C}}(s\mathbf{I} - \hat{\mathbf{A}})^{-1}\hat{\mathbf{B}} + \mathbf{D}.$$

Note that $m$, the order of the intermediate reduced-order model $\tilde{\Sigma}$ as introduced in Step 5, should be chosen such that it is feasible to compute all the finite stable generalized eigenvalues of $(\mathcal{A}_2, \mathcal{E}_2)$ and their residues.

**4. Software Implementation.** The two model-order-reduction algorithms have been implemented in C++ using the following Trilinos packages [3]: Epetra, EpetraExt, Amesos, Teuchos and Anasazi. In this section, for each of the methods, we first present necessary modifications or additions to the algorithm which extend its application (as in the SPRIM case) or make it suitable for the implementation (as in the case of the invariant subspace method). The modifications and additions are then followed by a detailed implementation description.

## 4.1. SPRIM.

**4.1.1. Modifications.** First, notice that the resulting MNA equations derived in the SPRIM algorithm as presented in Section 3.1 are for circuits excited only by current sources. Thus, for circuits excited by different types of sources other than only current ones, the block matrices in (3.3) need be reformulated accordingly.

Second, the transfer functions of a circuit and of its reduced model appear as shown in (3.5) and (3.6) because the outputs of the circuit are the node voltages observed at the same locations/ports as its inputs. If this is not the case, then consider the following addition to the algorithm:

Let $\mathcal{D}$ be a matrix of size $p \times q$, where $p$ and $q$ are the numbers of outputs and of states of the circuit, respectively. Each row of $\mathcal{D}$ represents an output and contains $+1$ in the column corresponding to the location of the interested output in the state vector $x$ as in (3.3), and 0 everywhere else. As part of Step 1 in Algorithm 1, partition $\mathcal{D}$ such that $\mathcal{D} = [\mathbf{D}_1 \ \mathbf{D}_2]$ in accordance with the block structure of $\mathcal{G}$ as in (3.3). In Step 5 of the same algorithm, also compute $\tilde{\mathbf{D}}_1 = \mathbf{D}_1\mathbf{V}_1$, $\tilde{\mathbf{D}}_2 = \mathbf{D}_1\mathbf{V}_2$ and form $\tilde{\mathcal{D}}_n = [\tilde{\mathbf{D}}_1 \ \tilde{\mathbf{D}}_2]$. Then the transfer functions of the circuit of interest and of its reduced model are respectively as follows,

$$\begin{aligned} \mathbf{G}(s) &= \mathcal{D}(\mathcal{G} + s\mathcal{C})^{-1}\mathcal{B}, \\ \tilde{\mathbf{G}}_n(s) &= \tilde{\mathcal{D}}_n(\tilde{\mathcal{G}}_n + s\tilde{\mathcal{C}}_n)^{-1}\tilde{\mathcal{B}}_n. \end{aligned} \tag{4.1}$$

**4.1.2. Implementation.** The following is a stepwise C++ implementation of Algorithm 1 using Trilinos packages [3]: In Step 1, Epetra and EpetraExt are used to read in matrices from files in Matrix Market format [1] and to form the matrices and corresponding block matrices accordingly. In Step 2, Amesos is used to perform two direct sparse linear solves. In Step 3, Anasazi is used to construct the Krylov subspace of interest. The computations and formations of the matrices in Steps 4 and 5 are performed via Epetra and EpetraExt with an extensive help of Teuchos. Note that Teuchos is used throughout the implementation as it provides convenient tools such as BLAS/LAPACK wrappers and smart pointers and that during the implementation, we exercise all the modifications/additions presented above.

## 4.2. Invariant Subspace.

**4.2.1. Modifications.** Notice that the non-zero finite generalized eigenvalues of $(\mathcal{A}, \mathcal{E})$ are the reciprocals of the non-zero finite eigenvalues of $\mathcal{A}^{-1}\mathcal{E}$ given that $\mathcal{A}$ is non-singular. Thus, mathematically, generalized eigenvalues with smallest modulus of $(\mathcal{A}, \mathcal{E})$ are the same as eigenvalues with largest modulus of $\mathcal{A}^{-1}\mathcal{E}$.

With this observation, for Step 2 in Algorithm 3, in our implementation, we compute an $m^{th}$-order partial real Schur decomposition $\mathcal{A}_1^{-1}\mathcal{E}_1\mathbf{Q} = \mathbf{Q}\mathbf{R}$ corresponding to the subset of $m$ non-zero stable eigenvalues with largest modulus of $\mathcal{A}_1^{-1}\mathcal{E}_1$ instead. Also, in Step 7 of the same algorithm, we compute all the non-zero stable eigenvalues of $\mathcal{A}_2^{-1}\mathcal{E}_2$ instead and then consider their reciprocals.

**4.2.2. Implementation.** The following is a stepwise C++ implementation of Algorithm 3 using Trilinos packages [3]: In Step 1, Epetra and EpetraExt are used to read in matrices from files in Matrix Market format [1] and to form the matrices and corresponding block matrices accordingly. Prior to Step 2, Amesos is used to perform a direct sparse linear solve as mentioned in Section 4.2.1. In Step 2, Anasazi is used to construct a partial real Schur decomposition $\mathcal{A}_1^{-1}\mathcal{E}_1\mathbf{Q} = \mathbf{QR}$. The computations and formations of the matrices in Steps $3-5$ are performed via Epetra and EpetraExt with an extensive help of Teuchos. In Step 6, Epetra and EpetraExt are used again to form the block matrices. Prior to Step 7, Epetra is used to perform a matrix inversion as mentioned in Section 4.2.1. In Step 7, a few simple functions are written to compute the reciprocals of the eigenvalues of $\mathcal{A}_2^{-1}\mathcal{E}_2$ and their residues. Epetra and EpetraExt with an extensive help of Teuchos are used for the remaining steps to compute and form the matrices. Also, as mentioned in the SPRIM case, Teuchos is used extensively throughout the implementation, and the modifications above are applied.

**5. Experimental Results.** In this section, we present numerical results from running the two implemented passivity preserving MOR methods on an RLC circuit system. The frequency responses of the original and reduced models are presented to demonstrate how good approximations the reduced models are, compared to the original. For the performance of the MOR methods with respect to the order of the reduced model, see the corresponding works in Freund [2] and Nong [5].

At the current stage of the project, RLC systems are extracted from Xyce in the form of data files in Matrix Market format [1]. These files will be the inputs to the C++ code for the SPRIM method. For the invariant subspace method, since the algorithm only considers non-descriptor systems for the time-being, these data files need to be further processed so that the state space realizations of the systems are put in non-descriptor form. In other words, at the moment, with any given model we are using two different generators to generate both a descriptor and non-descriptor form of the dynamical system.

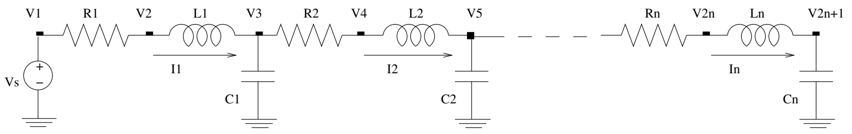
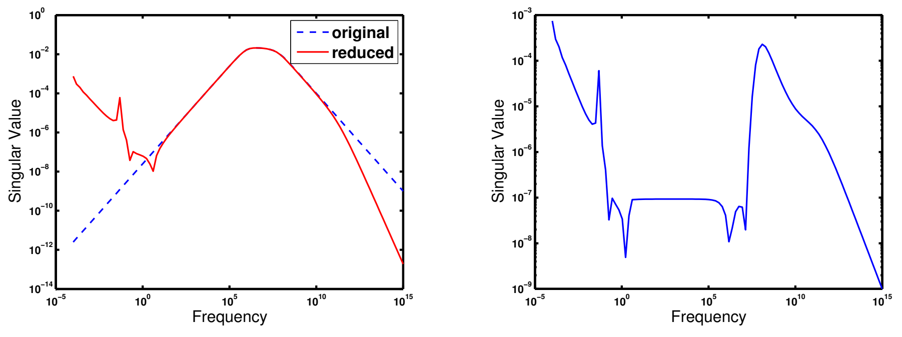


FIG. 5.1. *An RLC ladder circuit.*

**5.1. Test Model.** We consider an RLC ladder circuit excited by a voltage source as depicted in Figure 5.1. RLC ladder circuits of this type are very typically used to model the interconnect between the devices on electronic chips. The circuit model we are working with has 50 blocks of RLC cells, i.e., $n = 50$. Note from Figure 5.1 that in each block $i$, the node voltage $V_{2i}$ between the resistor $R_i$ and inductor $L_i$ is a state. Therefore, for this circuit, an MNA formulation results in an input LTI model of size 151 for the SPRIM algorithm. For the invariant subspace method, further processing produces a non-descriptor input LTI model with a state space realization of order 100. The numerical values for the resistor, inductor and capacitor are $R_i = 0.2\Omega,\ L_i = 1\mu H,\ C_i = 0.5nF$, for $i = 1, \ldots, n$.

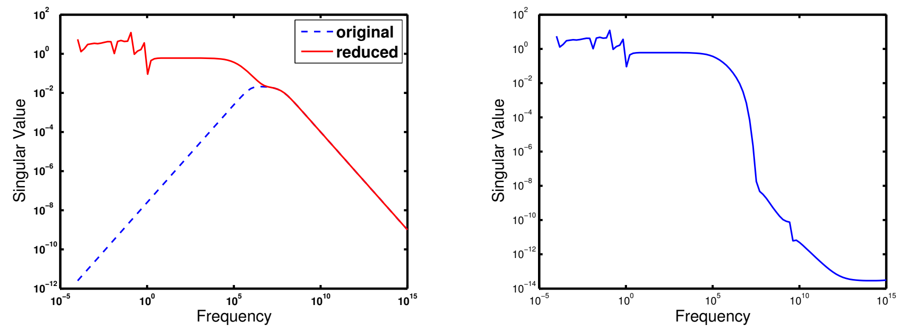
**5.2. Results.** Given the RLC ladder circuit in Section 5.1, we are interested in constructing reduced models of order 40 and 20 using the SPRIM and invariant subspace algorithms, respectively. Note that, even though the descriptor and non-descriptor form of the dynamical system constructed for the two MOR methods (SPRIM and invariant subspace) model the same circuit, they are of different orders (151 and 100, respectively) due to different generators. Thus, it is not quite obvious how to select the reduced orders so that the two corresponding reduced models of the two original systems are comparable. As it appears reasonable, we choose 40 (with respect to 151) and 20 (with respect to 100) as the orders of the corresponding reduced systems resulting from model reduction using the two associated methods.

Since the Krylov subspace in the SPRIM algorithm depends on the expansion point as an input parameter, we present here two cases for the SPRIM algorithm in which the expansion points are chosen to be 1000 and $10^9$. Figures 5.2 and 5.3 present the frequency responses of the original and reduced models resulting from applying the SPRIM method, while Figure 5.4 that from the invariant subspace method.



(a) (b)

Fig. 5.2. *Frequency responses of (a) the original model (as described in Section 5.1) of order* 151 *and its reduced model of order* 40 *and (b) their error system resulting from applying the SPRIM algorithm about an expansion point at* 1000.



(a) (b)

Fig. 5.3. *Frequency responses of (a) the original model (as described in Section 5.1) of order* 151 *and its reduced model of order* 40 *and (b) their error system resulting from applying the SPRIM algorithm about an expansion point at* $10^9$.

**5.3. Observations.** For the SPRIM algorithm, the expansion points need to be specified a priori. The behavior of the reduced models appears local. In other words, given an original model and an expansion point, the reduced model will capture the features of the original in the frequency range about the pre-specified expansion point.

For the invariant subspace algorithm via the two-stage reduction method, the reduced models appear to capture the features of the original in a global sense. This in fact agrees

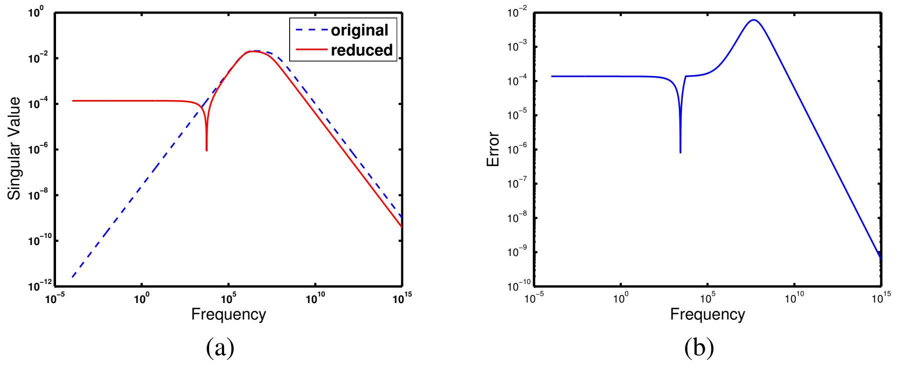


Fig. 5.4. *Frequency responses of (a) the original model (as described in Section 5.1) of order* 100 *and its reduced model of order* 20 *and (b) their error system resulting from applying the invariant subspace algorithm via the two-stage reduction method.*

with what Nong describes in [5]: Spectral zeros with largest residue are the most important in terms of energy components. Thus, interpolation over these terms results in a reduced model that captures most of the energy of the original dynamical system.

## 6. Conclusions & Future Work.

**6.1. Conclusions.** We present a preliminary study of the infrastructure required for the integration of current MOR methods into Xyce. To this end, we implement two passivity preserving MOR methods for LTI systems in C++ using Trilinos [3]. The two methods are the SPRIM algorithm proposed by Freund [2] and the invariant subspace algorithm proposed by Sorensen [7] and further studied by Nong [5]. In addition to the implementation, we conclude through experimental results that, compared to the original model, the SPRIM algorithm produces reduced models that are good approximations in a local sense while reduced models resulting from the invariant subspace algorithm via the two-stage reduction method are good approximations in a global sense.

**6.2. Future Work.** At the current stage of the project, the implementation remains a separate entity with respect to Xyce ; inputs for the implemented codes are obtained via data files that are extracted from Xyce . In addition, for the invariant subspace method that requires inputs in non-descriptor form, the extracted information needs to be further processed. A few observations based on this issue are that not all descriptor systems can be modified to put in non-descriptor form and that when the modifications are possible, they are not universal, i.e., the modifications are system-dependent. For the SPRIM algorithm, knowledge of expansion points is too subjective, and for a few cases, the frequency range for assuring a good approximation does not always enclose the expansion point. Moreover, the circuit structure that the SPRIM method preserves may not be known a priori in modern circuit simulators, such as Xyce, since these simulators often use modern software design techniques resulting in an abstract notion of devices. These limitations suggest the following future directions for the project:

1. Examine if the invariant subspace method can be generalized for descriptor systems. If it is possible, then work on providing a generalization.
2. For the SPRIM algorithm, study in detail the role of expansion points in the formation of reduced models and examine if it is possible to construct a reduced model about multiple expansion points to better capture the behavior of the original system

in a global sense. In addition, examine the structure preservation techniques and, if it is possible, study how to abstract these structure preservation techniques.

3. Introduce criteria on what method is suitable for which LTI system and implement the entire code as a black box embedded inside Xyce.

In addition, so far we have only considered MOR techniques for LTI circuits. Further steps for the project would also include investigating the MOR techniques for nonlinear and time variant circuits. The ultimate goal of this work is to enable intelligent integration of MOR methods into Xyce and expedite the full system simulation of very highly integrated circuits.

## REFERENCES

[1] *Matrix market*. `http://math.nist.gov/MatrixMarket`, May 2007.

[2] R. W. Freund, *SPRIM: Structure-preserving reduced-order interconnect macromodeling*, Proceedings of the 2004 IEEE/ACM International Conference on Computer-Aided Design, (2004), pp. 80–87.

[3] M. A. Heroux, R. A. Bartlett, V. E. Howle, R. J. Hoekstra, J. J. Hu, T. G. Kolda, R. B. Lehoucq, K. R. Long, R. P. Pawlowski, E. T. Phipps, A. G. Salinger, H. K. Thornquist, R. S. Tuminaro, J. M. Willenbring, A. Williams, and K. S. Stanley, *An overview of the Trilinos project*, ACM Transactions on Mathematical Software, 31 (2005), pp. 397–423.

[4] E. R. Keiter, S. A. Hutchinson, R. J. Hoekstra, T. V. Russo, and L. J. Waters, *Xyce™ parallel electronic simulator design: Mathematical formulation. Version 2.0*, SAND2004-2283, Sandia National Laboratories, 2004.

[5] H. D. Nong, *Passivity preserving model reduction via interpolation of spectral zeros: Selection criteria and implementation*, master's thesis, Rice University, 2007.

[6] A. Odabasioglu, M. Celik, and L. T. Pileggi, *PRIMA: Passive reduced-order interconnect macromodeling algorithm*, IEEE Transactions on Computer-Aided Design of Intergrated Circuits and Systems, 17 (1998), pp. 645–654.

[7] D. C. Sorensen, *Passivity preserving model reduction via interpolation of spectral zeros*, Systems & Control Letters, 54 (2005), pp. 347–360.

# HESSIAN-BASED MODEL REDUCTION APPROACH TO SOLVING LARGE-SCALE SOURCE INVERSION PROBLEMS

CHAD E. LIEBERMAN[*] AND BART G. VAN BLOEMEN WAANDERS[†]

**Abstract.** The source inversion problem in the application of contamination requires methods for real-time computation. Determination of the location and magnitude of the contaminant source given sparse sensor data is formulated as an optimization problem constrained by the system dynamics. The computational complexity of large-scale finite element discretizations precludes real-time inversion by any method. To effect real-time computation, we utilize Hessian-based model reduction to produce an optimal reduced-order model sensitive to all initial conditions. Through an application to the convection-diffusion equation in a two-dimensional domain, we demonstrate the success of our inversion algorithm. The improvement in computation time over the full-order inversion is quantified. A high level abstraction toolkit is leveraged to efficiently implement the dynamics and extract linear operators for components of the Hessian.

## 1. Introduction.

The source of contamination events characterized by sparse sensor information is obtained by minimizing the misfit between observations and numerical predictions. While chemical spills, gas leaks, and groundwater contamination are example applications where source inversion has been applied, chemical, biological, and radiological terrorist attacks have emerged as significant threats. In principle, source inversions for any contamination event share similar algorithmic characteristics – i.e. large number of inversion parameters, use of observations, regularized objective function – but in the case of a terrorist attack scenario, not only are the location and magnitude unknown, the inversion needs to be conducted in real time. This adds a significant challenge to algorithmic development. Furthermore, the ability of the algorithms to accurately predict the character and location of the original source is integral to the support of potential evacuation procedures as well as the mitigation process of the hazardous effects.

Inverse problems of this form have been extensively studied. The literature encompasses methods in stochastic estimation and deterministic optimization. In either case, research in this field is dominated by inversion for a relatively small set of parameters. In contrast, we are interested in inverse problems for which the number of inversion variables is equivalent to the number of degrees of freedom in the computational domain. In addition, we are interested in algorithms capable of real time efficiency. In contamination applications the finite element discretization of the domain often result in millions of degrees of freedom. Inverse computation at full-order could require days of run time even if implemented in parallel on today's fastest supercomputers. With the advent of model reduction, we can reduce the computation time by orders of magnitude and thereby approach real-time analysis. While some researchers have recognized the need for reduced-order models in inverse problems [6, 9], their model reduction approaches do not specifically target the inversion. Instead, reduced-order models are used to decrease the computation time of forward solves required by each iteration of the solution. Hessian-based model reduction produces an optimal reduced-order model designed for a deterministic optimization formulation of the source inversion problem.

There are several approaches to solving source inversion problems. Probabilistic methods build uncertainty estimation directly into the model [3, 12, 14]. A use of the particle method results in a set of probability distributions for initial conditions [3]. The inversion leads to the most probable source and automatically provides quantification of the uncertainty associated with the solution. Snodgrass et. al. combine Bayesian theory and geostatistical techniques to invert for a time-dependent groundwater pollution source [14]. The Bayesian

[*]Massachusetts Institute of Technology, celieber@mit.edu

[†]Sandia National Laboratories, bartv@sandia.gov

approach leads directly to error quantification and provides information about the source of uncertainty in the prediction. Geostatistical techniques utilize the information about uncertainty to adaptively improve the model, thereby making the algorithm applicable to a general set of release history problems. An extended Kalman filter was applied to the groundwater inverse problem in Ref. [12]. The Kalman filter builds up an estimate of the state at each timestep as more information is collected by sensors. In addition to probabilistic approaches, deterministic optimization is also well represented in the inverse problem literature. The formulations usually include a combination of misfit minimizations and regularization. The research in this sector encompasses experiments in regularization and improvements in computation of the resulting linear systems. Multigrid preconditioners were investigated in Ref. [1]. A preconditioner is constructed to force the eigenvalues into clusters, thereby improving the performance of an iterative solver. In Ref. [2], an improvement on the conjugate gradients iteration by exploitation of the inverse operator's spectral structure permitted faster computation. In the present article, we formulate the inverse problem as an optimization constrained by system dynamics. To achieve real time efficiency, we incorporate a specialized model reduction algorithm.

Model reduction and optimization have been united in two ways in the literature. Firstly, the creation of a reduced-order model results from the minimization of differences in characteristics between the reduced-order and full-order systems. Secondly, a reduced-order model is used as one element in an optimization algorithm. In the former case, researchers are concerned with matching the reduced-order and full-order outputs or transfer functions [4, 7, 18, 11, 15]. While Refs. [4, 7, 18] use a goal-oriented approach to focus the optimization around minimizing the $L_2$ error between full-order and reduced-order outputs, Ref. [15] constructs a reduced-order model based on a relaxation of the $H_\infty$ norm. In Ref. [11], the reduced-order model is focused on minimizing the deviations between the frequency responses of the full-order and reduced-order systems over the range of interest. Researchers also use reduced-order models to make tractable some large-scale optimization problems. In these cases, the reduced-order models are substituted for full-order finite element solutions. For example, optimal control problems are solved with reduced-order models [5, 13, 16]. Bergmann et. al. use a modification of proper orthogonal decomposition to develop a reduced-order model to decrease the run time for each computational fluid dynamics calculation involved in minimizing the drag on a rotating cylinder. A reduced-order model is created for optimizing the set of power inputs for the lamp banks in a problem of rapid thermal chemical vapor deposition in the field of semiconductor manufacturing [16]. The present work combines these applications of model reduction and optimization by utilizing Hessian-based model reduction to solve the source inversion problem.

The inverse problem is often posed as an optimization problem; therefore, a reduced-order model may be effective in this context as well. Reduced-basis methods are utilized in the inverse identification of thermal parameters of a microelectronics package [9]. A projection basis spanning a set of full-order finite element solutions is used to project the system dynamics to reduced space. The computation time is reduced further by implementing a combination of genetic algorithms and hillclimbing techniques to reduce the number of forward solves required by the optimization. A Krylov subspace model reduction technique is applied to the inverse problem of electromagnetic scattering [6]. The reduced-order model, based on the shift invariance property, is developed as part of the Lanczos or Arnoldi algorithms used to solve for the scatterer electric field. In this article we use an optimal reduced-order model to minimize the $L_2$ error between the simulated time evolution of the initial condition and the sparse sensor data.

This class of inverse problems may be addressed by a measure-invert-predict-control methodology. The first step is to measure the concentration of the contaminant in space and time. Sensors placed in the domain collect readings of the local concentration of the contaminant. Next, the data from these sensors are used to solve an optimization problem whose solution is the initial contaminant release. Information about the location and magnitude of the source of the contaminant is integral to the emergency decision-making process. However, in many cases, knowledge of the source is not sufficient: prediction of the forward-time propagation of the contaminant is required to conduct appropriate evacuations. If the contaminant can be controlled, then the predictions may be used to develop an optimal control strategy. Control actuators depend on the nature of the contaminant release and the governing equations. In some cases, contaminants can be neutralized by a chemical agent. One control strategy would command judicious release of the neutralizing agent in regions of high contaminant concentration. If the contaminant is released indoors, the heating, ventilation, and air-conditioning (HVAC) system may be used to evacuate the chemical. Vents may be used to force the contaminant to a specified location while returns are used to expel it. The control problem is not addressed here but it presents a challenge in future work.

This articles addresses the task of obtaining real-time solutions to large-scale source inversion problems utilizing Hessian-based model reduction. It is structured as follows. In Section 2 we consider the mathematical formulation of the optimization problem whose solution is the contaminant initial condition. The algorithm is first derived at full-scale. Then, in Section 3, we briefly discuss Hessian-based model reduction and how it is used to solve the same problem in reduced space. Section 4 presents implementation details and highlights our work with a rapid-development finite element toolkit. In Section 5 the full-order and reduced-order inversion schemes are applied to a generic contaminant release governed by the convection-diffusion equation. The results and analysis are presented in Section 6. A qualitative, preliminary parameter study is found in Section 7. Finally, we present conclusions in Section 8.

## 2. Formulation.

The goal of source inversion is to predict initial conditions that, when simulated forward in time, produce dynamics consistent with the sparse sensor data. Therefore, we seek to minimize the difference between the actual and predicted concentrations at the sensor locations. A regularization term is included in the objective function to avoid illposedness.

Given sensor readings in space-time $\mathbf{y}^*$ and Tikhonov regularization parameter $\beta$, the inversion problem is written as a constrained optimization:

$$\min_{\mathbf{x},x_0} \mathcal{J}(\mathbf{x}, x_0) = \frac{1}{2}(\mathbf{y}-\mathbf{y}^*)^T(\mathbf{y}-\mathbf{y}^*) + \frac{\beta}{2}x_0^T x_0$$

$$\text{subject to} \quad \mathbf{A}\mathbf{x} = \mathbf{F}x_0, \tag{2.1}$$

$$\mathbf{y} = \mathbf{C}\mathbf{x}, \tag{2.2}$$

where Equations (2.1)-(2.2) are the general linear discrete space-time equations resulting from stacking up the equations of the discrete-time system

$$x(k+1) = Ax(k), \quad k = 0, 1, \dots, T-1, \tag{2.3}$$

$$y(k) = Cx(k), \quad k = 0, 1, \dots, T, \tag{2.4}$$

$$x(0) = x_0, \tag{2.5}$$

for $k = 0, 1, \dots, T$. In Equations (2.3)-(2.5), $x(k) \in \mathbb{R}^N$ is the system state, $x_0$ is the initial condition, and $y(k) \in \mathbb{R}^Q$ is the output. The matrices $A \in \mathbb{R}^{N\times N}$, $B \in \mathbb{R}^{N\times P}$, and $C \in \mathbb{R}^{Q\times N}$

result from the choices of spatial and temporal discretization. In general, systems of this form will arise from finite element discretizations of complex spatial domains. For backward Euler temporal discretization, the matrices in Equations (2.1)-(2.2) contain the following structure:

$$\mathbf{x} = \begin{bmatrix} x(0) \\ x(1) \\ \vdots \\ x(T) \end{bmatrix}, \quad \mathbf{y} = \begin{bmatrix} y(0) \\ y(1) \\ \vdots \\ y(T) \end{bmatrix}, \quad \mathbf{F} = \begin{bmatrix} I \\ 0 \\ 0 \\ \vdots \\ \vdots \\ 0 \end{bmatrix},$$

$$\mathbf{A} = \begin{bmatrix} I & 0 & 0 & & \cdots & 0 \\ -A & I & 0 & \ddots & & \vdots \\ 0 & -A & I & \ddots & \ddots & \\ & \ddots & \ddots & \ddots & \ddots & 0 \\ \vdots & & \ddots & \ddots & \ddots & 0 \\ 0 & \cdots & & 0 & -A & I \end{bmatrix}, \quad \mathbf{C} = \begin{bmatrix} C & 0 & \cdots & & \cdots & 0 \\ 0 & C & 0 & & & \vdots \\ \vdots & 0 & C & \ddots & & \\ & & \ddots & \ddots & \ddots & \vdots \\ \vdots & & & \ddots & \ddots & 0 \\ 0 & \cdots & & \cdots & 0 & C \end{bmatrix}.$$

Introducing the adjoint variable $\mathbf{z} \in \mathbb{R}^{NT}$, we form the Lagrangian functional $\mathcal{L}(\mathbf{x}, x_0, \mathbf{z}) = \mathcal{J}(\mathbf{x}, x_0) - \mathbf{z}^T(\mathbf{y} - \mathbf{C}\mathbf{A}^{-1}\mathbf{F}x_0)$. Requiring stationarity of the first variations of $\mathcal{L}$ with respect to the state variable $\mathbf{x}$, the input $x_0$, and the co-state $\mathbf{z}$ yield the adjoint equation, the optimality condition, and the constraint equation, respectively. Let $\mathbf{g}$ be the first variation of $\mathcal{L}$ with respect to the input $x_0$. Stationarity of $\mathcal{L}$ is then given by

$$\mathbf{g} = (\mathbf{H} + \beta\mathbf{I})x_0 - (\mathbf{C}\mathbf{A}^{-1}\mathbf{F})^T\mathbf{y}^*,$$

where $\mathbf{H} = (\mathbf{C}\mathbf{A}^{-1}\mathbf{F})^T(\mathbf{C}\mathbf{A}^{-1}\mathbf{F})$ is the Hessian and $\mathbf{I}$ is the identity matrix of dimension $N$. Therefore, the initial condition that minimizes the objective function is the solution to the linear problem

$$(\mathbf{H} + \beta\mathbf{I})x_0 = (\mathbf{C}\mathbf{A}^{-1}\mathbf{F})^T\mathbf{y}^*. \tag{2.6}$$

Although this linear system of dimension $N$ is already very large for finite element applications, the true computational complexity is disguised within the Hessian operator $\mathbf{H}$. In a large-scale application, the matrices composing $\mathbf{H}$ cannot be formed; instead, they must be defined by their action on a vector. The reduction of order of these operators is the motivation for implementing Hessian-based model reduction.

**3. Hessian-based model reduction.** Many model reduction techniques are methods to find a reduced-space basis $V \in \mathbb{R}^{N\times n}$ with $n \ll N$ by which the general discrete-time system (2.3)–(2.5) is projected to yield

$$\begin{aligned} x_r(k+1) &= A_r x_r(k), \quad k = 0, 1, \ldots, T-1, \\ y_r(k) &= C_r x_r(k), \quad k = 0, 1, \ldots, T, \\ x_r(0) &= V^T x_0, \end{aligned}$$

where $A_r = V^T A V$, $C_r = CV$, and $x_r = V^T\hat{x}$ is the reduced state expansion in the basis.

The Hessian-based model reduction strategy seeks a reduced-space basis that minimizes the difference in outputs between the full-order and reduced-order systems. In Ref. [4], it is shown that the optimal basis can be obtained by coupling proper orthogonal decomposition (POD) with the greedy sampling approach originally proposed in Ref. [17]. Bashir et. al. find that the dominant eigenvector of the error Hessian $\mathbf{H}^e = (\mathbf{C}\mathbf{A}^{-1}\mathbf{F} - \mathbf{C}_r\mathbf{A}_r^{-1}\mathbf{F}_r)^T(\mathbf{C}\mathbf{A}^{-1}\mathbf{F} - \mathbf{C}_r\mathbf{A}_r^{-1}\mathbf{F}_r)$ is the initial condition that maximizes the $L_2$ error between reduced-order and full-order outputs. One way to obtain an optimal reduced-order basis would be to initialize empty reduced-order model matrices, sample the dominant eigenvector of $\mathbf{H}^e$, construct a reduced-order model, sample the dominant eigenvector of $\mathbf{H}^e$, etc. However, instead of repeatedly sampling the dominant eigenvector of the error Hessian and constructing a new reduced-order model at every iteration, an arbitrary initial condition may be rewritten in terms of its components in the space of the reduced basis and those orthogonal to that basis. The simplifying assumption that the reduced-order output exactly matches the full-order output for initial conditions in the basis permits a one-shot method for snapshot collection. Once an eigenvector cutoff criterion is selected, the dominant eigenvectors satisfying that criterion are each used as seed initial conditions in a forward solve. This process can be parallelized by running the forward solves over the number of available processors. Snapshots of the state are taken from the resulting time evolution and POD is performed to obtain a reduced-order basis. For a more complete discussion of the algorithm, please see Ref. [4].

Once the reduced-order basis is computed, the full-scale system may be projected onto the reduced space. The fully discrete system (2.1)–(2.2) is rewritten as

$$\begin{aligned}\mathbf{A}_r\mathbf{x}_r &= \mathbf{F}_r x_0,\\ \mathbf{y}_r &= \mathbf{C}_r\mathbf{x}_r,\end{aligned}$$

where the reduced space-time matrices contain the appropriately projected submatrices. The reduction to $\mathbf{A}_r \in \mathbb{R}^{nT\times nT}$, $\mathbf{F}_r \in \mathbb{R}^{nT\times N}$, and $\mathbf{C}_r \in \mathbb{R}^{N\times nT}$ results in a decrease of many orders of magnitude in the number of floating point operations (flops) required by a solver iteration. The only requirement of the Hessian-based model reduction technique is full knowledge of the governing equations. If the equations are known, calculation of the reduced-order model, the most computationally intensive aspect of our source inversion methodology, can be pre-computed offline. The source inversion then boils down to solving the reduced linear problem

$$(\mathbf{H}_r + \beta\mathbf{I})x_0 = (\mathbf{C}_r\mathbf{A}_r^{-1}\mathbf{F})^T\mathbf{y}^* \tag{3.1}$$

where $\mathbf{H}_r = (\mathbf{C}_r\mathbf{A}_r^{-1}\mathbf{F})^T(\mathbf{C}_r\mathbf{A}_r^{-1}\mathbf{F})$. With the reduction in computation time, Equation (3.1) may be solved in real-time on a laptop computer in the field.

**4. Implementation.** Algorithms that require access to the underlying linear algebra infrastructure of the target dynamics pose significant implementation challenges, especially for complex physics, finite element discretizations, and parallelization. In this work we avoid development costs associated with the low-level implementation of the forward simulation code by leveraging high level abstraction methods. The general concept is to isolate implementation requirements from the end-user to allow exclusive focus on the physics formulation. Sandia's Sundance [10] is utilized in this context and provides the capability to write sets of PDEs in the weak form for a finite element discretization. Differential and algebraic operators can be specified with test and unknown functions within the computational domain along with appropriate boundary conditions. The use of this tool merely requires a weak formulation for the specification of the physics (in addition to some other details which is handled by boiler plate code) through which a fully functional, 3D, parallel simulator is produced.

Although the convection-diffusion physics can be implemented trivially, the implementation of the reduced Hessian is not quite as straightforward. In our formulation, the resulting reduced Hessian is composed of multiple non-trivial linear operators. First, the $\mathbf{A}$ matrix represents the complete time-stepping sequence of the dynamics, which in this case is convection-diffusion. Then the mapping of initial conditions to time and space is required through the matrix $\mathbf{F}$, followed by a matrix $\mathbf{C}$ to identify the appropriate sensor locations. To complete the Hessian construction, a transpose of $\mathbf{CA}^{-1}\mathbf{F}$ is also needed. Although, the transpose is not a complicated computation to implement, in parallel this can be non-trivial and not all linear algebra infrastructures have this capability. Finally, the Hessian is dense, large, and cannot be formed for any reasonable size dataset. These operations have to be implemented by defining the actions of matrices on vectors through linear operators and corresponding apply methods. These requirements are not immediately accessible at the end-user level but the underlying Sundance linear algebra infrastructure is based on matrix free solution procedures. Therefore a special abstract base class was created to build operators related to reduced order models (ROMs) for linear time-invariant (LTI) systems. An Abstract Factory Pattern [8] provides the appropriate encapsulation for the LTI components and other ROM related linear objects. The convection-diffusion main program creates a concrete implementation of the abstract factory which creates the concrete objects from the interfaces, in this case all the constituents of $\mathbf{H}$.

A simple 2-D computational domain was used to demonstrate our algorithm and was defined directly within Sundance using the `BasicSimplicialMesher` method. The convection-diffusion equation was defined in the weak form using the Dirichlet boundary condition on the left boundary. Using linear elements, Sundance constructs a mesh of $N$ nodes. The timestepping matrix $A$ was then extracted from the linear problem setup by the `getOperator()` method. The space-time $\mathbf{A}^{-1}$ operator was implemented as a series of matrix-vector products utilizing the number of time steps $T$. After reading in the specified sensor locations, the matrix $\mathbf{C}$ was defined using Thyra[1] multivectors. The Hessian operator is composed of its constituents and passed to Matlab. After completion of the eigenvector analysis, the seed initial conditions are loaded back into Sundance where a sequence of forward solves generate the snapshot matrix.

The Matlab[2] `eig` function is utilized to generate the dominant eigenvectors of the Hessian imported from Sundance. Eigenvectors are selected as initial conditions if their associated eigenvalues satisfy $\lambda_j/\lambda_1 > \bar{\lambda}$ for user specified $\bar{\lambda}$. After they are passed back to Sundance and the snapshot matrix is generated, the `svd` command generates left singular vectors for the projection basis. Singular vectors are included in the projection basis if their normalized singular values satisfy $\sum_{i=1}^{j} \sigma_i < \text{POD}_{\text{crit}}$. Thus, the projection basis captures at least $\text{POD}_{\text{crit}} \times 100\%$ of the energy of the snapshot matrix. After the model matrices are projected and the reduced-order space-time system is generated, the initial condition is recovered by solving Equation (3.1) by Gaussian elimination. Eventually all of the Matlab functionality will be transferred to Sundance, thereby encapsulating all the functionality in one program. While the eigenvalue solver interface has been implemented, the projection and reduced-order inversion is still forthcoming.

Extension of numerical algorithms to large-scale computational domains is of great interest. Without a toolkit to rapidly test these algorithms for varying dynamics, parameters, and domains, many years are spent developing finite element codes for one-time use. Sundance provides the functionality to test these algorithms in parallel for different dynamics and complex domains. We next demonstrate our formulation and implementation using convection-

[1]Thyra is a Trilinos package. For more information, visit `http://trilinos.sandia.gov`.

[2]The MathWorks, Inc., Natick, MA 01760

diffusion dynamics but with relatively little effort our formulation can be tested on virtually any linear dynamics.

**5. Application to large scale inverse problem.** The convection-diffusion equation is solved on a simple 2-D rectangular domain to demonstrate the power of the Hessian-based model reduction technique. Although we would like to extend our implementation to large-scale 3-D problems in the future, the simple 2-D example here sufficiently demonstrates our algorithm. The extension to three dimensions only requires greater computation power in the construction of the reduced-order model. The formulation and the solution are directly analogous to the 2-D case. The governing equations are

$$\frac{\partial u}{\partial t} + \mathbf{v} \cdot \nabla u - \kappa \nabla^2 u = 0, \quad u \in \Omega, \tag{5.1}$$

$$u = 0, \quad u \in \Gamma_D, \tag{5.2}$$

$$\nabla u \cdot n = 0, \quad u \in \Gamma_N, \tag{5.3}$$

$$u = u_0, \quad u \in \Omega \times \{t = 0\}, \tag{5.4}$$

where $u$ is the concentration of the contaminant with initial condition $u_0$, $\mathbf{v} = \langle 2, 0 \rangle$ is the flow velocity, $\kappa$ is the diffusivity defined by $Pe = \|\mathbf{v}\|/\kappa$ with $Pe = 100$, and n is normal to the edge of the domain. There is a homogeneous Dirichlet boundary condition on the left of the domain and Neumann boundary conditions on the remainder of the perimeter as seen in Figure 5.1.

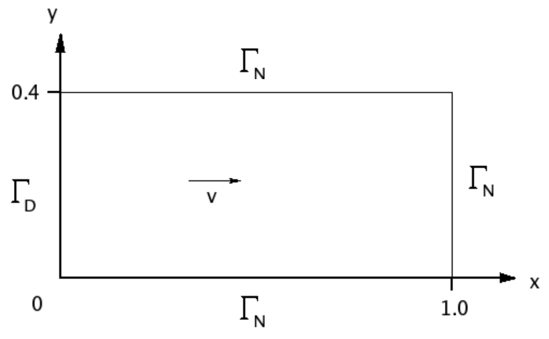


FIG. 5.1. *2-D computational domain with Dirichlet boundary condition on the left and Neumann boundary conditions on the remainder of the perimeter. The velocity field points uniformly in the x-direction throughout the domain.*

In order to test our inversion technique, we fabricate an initial condition and space-time evolution, thereby extracting sensor data at predetermined nodes of the computational domain. The initial condition is chosen to be a Gaussian in the center of the domain as pictured in Figure 5.2.

Forward-time simulations are achieved by backward Euler temporal discretization and finite element spatial discretization of Equation (5.1) using linear elements. The state is calculated at each time step of $\Delta t = 0.04$ for a total of $T = 10$ time steps. During this period, the contaminant diffuses and convects to the right side of the domain. The sensor sparsity is manifested within $\mathbf{C}$, and the sensor locations are overlayed on the domain in Figure 5.3.

For comparison, the source inversion is solved using the full-order system and the reduced-order model. Results and computational complexity are compared.

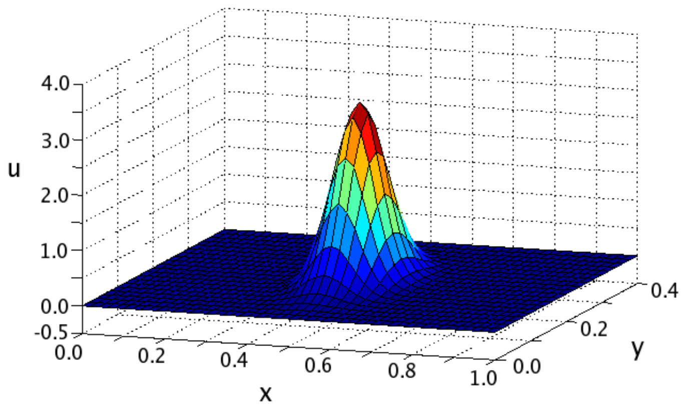


FIG. 5.2. *Actual initial condition. Gaussian placed in the center of the domain.*

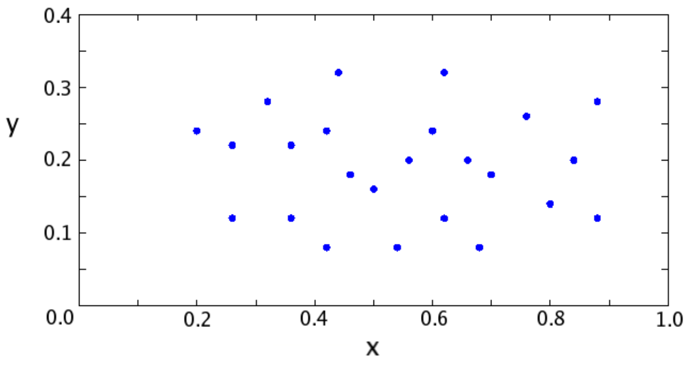
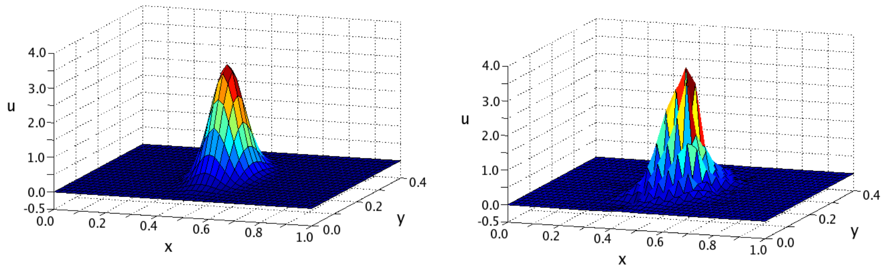


FIG. 5.3. *Sensor locations in the domain.*

**6. Results.** In this section the success of our reduced order algorithms is demonstrated. We find the initial condition that, when subject to regularization and simulated forward in time, most closely matches the sensor measurements in a least squares sense. In the discrete form, the optimization problem reduces to a linear problem, the solution of which requires the application of large space-time operators at each iteration. Hessian-based model reduction reduces the size of these operators while maintaining the integrity of the input-output relations corresponding to the initial condition space and the sensor locations. While the full-order system may be solved in parallel on supercomputers, the model reduction results in a system solvable in serial on a laptop in the field in real time.

Figure 6.1 presents the full-order inversion side-by-side with the target initial condition. The height and footprint of the actual initial condition are nearly replicated. The only blemish is at the maximum concentration of the initial condition. In the full-order inversion, the concentration falls off somewhat more dramatically from the location of maximum concentration than does the smooth Gaussian of the target.

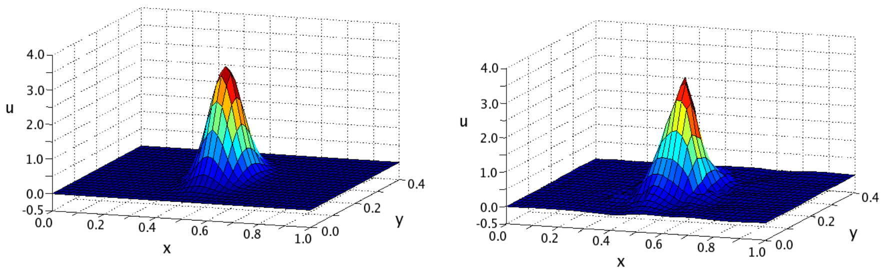
The reduced-order model inversion is pictured in Figure 6.2 with the target initial condition for comparison. Although this inversion was completed with orders of magnitude less computational complexity, the initial condition is still recovered. There are some undulations



(a) Target initial condition. (b) Full-order inversion.

FIG. 6.1. *Target initial condition (a) and full-order inversion (b).*

in the domain at the base of the Gaussian initial condition. These subtle inaccuracies are likely a result of the model reduction parameter choices as they are not seen in the full-order inversion above.



(a) Target initial condition. (b) Reduced-order inversion.

FIG. 6.2. *Target initial condition (a) and reduced-order inversion (b).*

Table 6.1 contains the parameters used in the implementation. The most important point to note is the reduction from $N = 1071$ nodes to $n = 128$ nodes by the reduced-order model. Considering the application of the Hessian operator to a vector, if $T > N/n$, the number of flops for the ROM inversion is $\mathcal{O}(n^2NT)$. Compare that complexity with the full-order inversion requiring $\mathcal{O}(N^3T)$ flops. The Hessian-based model reduction produces a reduced-order model used to successfully invert for initial conditions and reduces the number of flops required by a factor on the order of $(N/n)^2$. This massive decrease in computational expense may allow these inversion problems to be solved in real-time.

TABLE 6.1
*Parameter values*

| | |
|---|---|
| number of sensors $N_s$ | 24 |
| number of time steps $T$ | 10 |
| number of nodes $N$ | 1071 |
| reduced-order model size $n$ | 128 |
| regularization parameter $\beta$ | 0.001 |
| eigenvalue cutoff $\bar{\lambda}$ | 0.1 |
| singular value cutoff $\mathrm{POD}_{\mathrm{crit}}$ | 0.9999 |

Both inversions recover the initial condition within acceptable error bounds. The differences between the full-order and reduced-order inversion arise due to the quality of the reduced-order model. If the reduced-order model captures the dynamics of the system well enough, the inversion quality will be adequate. In implementation, however, there is a tradeoff between a more accurate solution and a faster computation time. The next section considers further variations in the parameters required for source inversion.

**7. Observations in parameter variation.** This section discusses the outcome of a small parameter study which was not meant to be exhaustive but rather qualitative in nature. Upon the completion of additional functionality in Sundance, a comprehensive sensitivity study will be conducted. At that time we can make more quantitative statements about the effects of parameter variations. For now four important parameter variations are discussed, two (number of sensors and regularization constant) in the context of inversion and two ($\bar{\lambda}$ and $\text{POD}_{\text{crit}}$) in the context of the reduced order modeling.

The number of sensors and their placement is a crucial parameter for importing the data that drives the inversion. For the high fidelity inversion case, we conclude that an insufficient number of sensors ($N_s < 0.01N$) causes significant inversion errors. Also, the placement of sensors is important to capture information along the convective streamlines, especially if only a small number of sensors is available. Of course without convection the inversion is not possible. Algorithmically, we would like to have sensor readings at every node in the domain. While obviously not practical in a field implementation, this strategy also presents some difficulty for the model reduction algorithm. There is no guarantee that the dominant eigenvectors of the Hessian will satisfy the boundary conditions of the forward problem. For example, in our application in Section 5, an eigenvector with nonzero component on the left boundary does not satisfy the homogeneous Dirichlet condition. In fact, in the computation, those elements would be treated as zeroes, which means the effective initial condition does not satisfy the Hessian eigenvalue problem. This result detracts from the integrity of the snapshots and the reduced-order model.

The regularization term directly affects the conditioning of the matrix $\mathbf{H} + \beta\mathbf{I}$ appearing in Equation (2.6) and in reduced-form in Equation (3.1). In implementation, a larger $\beta$ will produce a faster inverse solve, and likewise, a smaller $\beta$ will cause convergence problems. In formulation, the regularization term penalizes the $L_2$ norm of the initial condition in the objective function, thereby transforming the optimization problem to one that is well-posed. In theory and practice, $\beta$ should be decreased as far as the implementation can handle to achieve the highest quality result. However, small enough $\beta$ will produce an operator so ill-conditioned that the problem will be unsolvable.

Each of these parameters affects the inversion process regardless of the presence of reduced-order modelling. Without enough sensors and appropriate regularization, even a full-scale inversion will not yield adequate results. There are two parameters that affect the quality of the reduced-order inversion: $\bar{\lambda}$ and $\text{POD}_{\text{crit}}$. The initial conditions used for the snapshots are determined by $\bar{\lambda}$. Those eigenvectors corresponding to eigenvalues $\lambda_j$ of the Hessian satisfying $\lambda_j/\lambda_1 > \bar{\lambda}$ are used as initial conditions in a sequence of forward problems. The state vectors resulting from those forward problems form the snapshot matrix from which the basis is created. As $\bar{\lambda}$ approaches zero, more and more eigenvector initial conditions are utilized and the snapshot matrix grows. The increase in size results in longer run times due to the solution of more forward problems as well as the need to obtain the singular value decomposition of a larger matrix. In general, including more initial conditions from which to obtain snapshots will better capture the complete dynamics of the system leading to a more accurate and robust reduced-order model. Our selection of $\bar{\lambda} = 0.1$ reflects the recommendation in [4].

The second parameter to be adjusted in the reduced-order model is $\mathrm{POD}_{\mathrm{crit}}$, a cutoff parameter for singular values of the snapshot matrix determining how many basis vectors are retained for the projection. Let $\mathbf{w}_i$ represent the $i$th left singular vector of the snapshot matrix and $\sigma_i$ its corresponding normalized singular value. One interpretation of the singular value is that it represents the energy associated with the corresponding left singular vector in reconstructing the dynamics of the snapshot matrix. It is well known that the error in the $L_2$ norm of the basis projection is characterized by the largest singular value excluded by $\mathrm{POD}_{\mathrm{crit}}$. This is the motivation for retaining basis vectors $\mathbf{w}_j$ that satisfy $\sum_{i=1}^{j} \sigma_i < \mathrm{POD}_{\mathrm{crit}}$. In this form, $\mathrm{POD}_{\mathrm{crit}}$ represents the percentage of energy to be retained by the basis projection. Our implementation uses $\mathrm{POD}_{\mathrm{crit}} = 0.9999$ thereby capturing at least 99.99% of the energy in the first 128 basis vectors. For simple problems, often a very high $\mathrm{POD}_{\mathrm{crit}}$ leads to a small reduced-order model, but still captures a large portion of the energy.

**8. Conclusions and future work.** We have demonstrated the implementation of Hessian-based model reduction in a general source inversion problem. The model reduction procedure yields a set of reduced-order model matrices which significantly decreases the computational requirements to converge the inversion while still maintaining a high level of accuracy. Given sensor data, we can invert for initial conditions in real-time. To the best of our knowledge, this work represents the first instance of a directed sampling approach to model reduction for inversion. The Hessian-based model reduction specifically targets the initial conditions leading to maximal error between reduced-order and full-order models. The adaptation to the greedy algorithm samples these initial conditions at one time to form a basis, and subsequently generates a reduced-order model that is sensitive to the entire initial condition space. With a set of governing equations, a reduced-order model can be precomputed offline, leaving only a linear problem to solve for the inversion. In the future, we would like to explore 3-D problems with millions of nodes in the computational domain which requires the implementation of additional functionality within Sundance. Additionally, the rapid-development toolkit will aid in a complete parameter study. We will be able to test various sets of dynamics, types of initial conditions, and different parameter settings to quantitatively analyze the effects on the inversion algorithm. Finally, we will explore the control problem associated with a contaminant release. Depending on the application, our control strategy could be used with available actuators to mitigate the negative effects of a contamination event.

**Acknowledgements.** This work was partially supported by the Computer Science Research Institute at Sandia National Laboratories, Albuquerque, New Mexico. We thank our collaborators Professor Karen Willcox of MIT, Professor Kevin Long of Texas Tech University and Professor Omar Ghattas of the University of Texas at Austin for their contributions to this work.

## REFERENCES

[1] V. Akcelik, G. Biros, A. Draganescu, O. Ghattas, J. Hill, and B. van Bloemen Waanders, *Inversion of airborne contaminants in a regional model*, International Conference on Computational Science: Advancing Science through Computation, Reading, England, (2006).

[2] V. Akcelik, G. Biros, A. Draganescu, J. Hill, O. Ghattas, and B. van Bloemen Waanders, *Dynamic data-driven inversion for terascale simulations: real-time identification of airborne contaminants*, Proceedings of the 2005 ACM/IEEE Conference on Supercomputing, (2005), p. 43.

[3] A. Bagtzoglou, D. Dougherty, and A. Tompson, *Application of particle methods to reliable identification of groundwater pollution sources*, Water Resources Management, 6 (1992), pp. 15–23.

[4] O. Bashir, K. Willcox, O. Ghattas, B. van Bloemen Waanders, and J. Hill, *Hessian-based model reduction for large-scale systems with initial condition inputs*, Int. J. Numer. Meth. Engng, (2007).

[5] M. Bergmann, L. Cordier, and J.-P. Brancher, *Optimal rotary control of the cylinder wake using proper orthogonal decomposition reduced-order model*, Physics of Fluids, 17 (2005), pp. 1–20.

[6] N. Budko and R. Remis, *Electromagnetic inversion using a reduced-order three-dimensional homogeneous model*, Inverse Problems, 20 (2004), pp. S17–S26.

[7] T. Bui-Thanh, K. Willcox, O. Ghattas, and B. van Bloemen Waanders, *Goal-oriented, model-constrained optimization for reduction of large-scale systems*, Journal of Computational Physics, 224 (2007), pp. 880–896.

[8] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns*, Addison-Wesley, 2000.

[9] G. Liu, J. Lee, A. Patera, Z. Yang, and K. Lam, *Inverse identification of thermal parameters using reduced-basis method*, Computer Methods in Applied Mechanics and Engineering, 194 (2004), pp. 3090–3107.

[10] K. Long, *Large Scale PDE-Constrained Optimization*, Springer Lecture Notes in Computational Science and Engineering, 2003, ch. Sundance Rapid Prototyping Tool for Parallel PDE Optimization.

[11] R. Luus, *Optimization in model reduction*, International Journal of Control, 32 (1980), pp. 741–747.

[12] D. McLaughlin and L. Townley, *A reassessment of the groundwater inverse problem*, Water Resources Research, 32 (1996), pp. 1131–1161.

[13] S. Ravindran, *A reduced-order approach for optimal control of fluids using proper orthogonal decomposition*, International Journal for Numerical Methods in Fluids, 34 (2000), pp. 425–448.

[14] M. Snodgrass and P. Kitanidis, *A geostatistical approach to contaminant source identification*, Water Resources Research, 33 (1997), pp. 537–546.

[15] K. Sou, A. Megretski, and L. Daniel, *A quasi-convex optimization approach to parametrized model order reduction*, Annual ACM IEEE Design Automation Conference: Proceedings of the 42nd annual conference on design automation, San Diego, CA, (2005), pp. 933–938.

[16] A. Theodoropoulou, R. Adomaitis, and E. Zafiriou, *Model reduction for optimization of rapid thermal chemical vapor deposition systems*, IEEE Transactions on Semiconductor Manufacturing, 11 (1998), pp. 85–98.

[17] K. Veroy, C. Prud'homme, D. Rovas, and A. Patera, *A posteriori error bounds for reduced-basis approximation of parametrized noncoercive and nonlinear elliptic partial differential equations*, AIAA Paper 2003-3847, Proceedings of the 16th AIAA Computationoal Fluid Dynamics Conference, Orlando, FL, (2003).

[18] K. Willcox, O. Ghattas, B. van Bloemen Waanders, and B. Bader, *An optimization framework for goal-oriented, model-based reduction of large-scale systems*, 44th IEEE Conference on Decision and Control, Seville, Spain, (2005).

# IMPLEMENTING AND PROFILING OF A VARIABLE BLOCK MATRIX-MATRIX MULTIPLY IN ML

IAN KARLIN[‡] AND JONATHAN HU[§]

**Abstract.** This report discusses the implementation of a variable block matrix multiply within the multilevel preconditioning package **ML**. In general, the matrix-matrix kernel dominates the multigrid setup time. We discuss the advantages that a variable block multiply has over a point multiply. We then discuss refactoring key pieces of the matrix matrix multiply, and show the benefits via numerical experiments. Finally, we discuss future directions to make this fully available within a multigrid preconditioner. The resulting speedups from this work could prove beneficial to any application that produces block matrices and uses multigrid preconditioning.

**1. Introduction.** The repeated numerical solution of large, sparse, linear systems is central to many parallel simulations at Sandia. Within the linear solver, the choice of preconditioning method can have a tremendous impact on the convergence and runtime of the solver, and hence, the entire simulation.

For applications that give rise to symmetric positive definite linear systems, multigrid methods are often a good choice of preconditioner. At Sandia, the main multigrid preconditioning package is **ML** [4], part of the Trilinos solver framework [7]. **ML** provides a variety of *algebraic* multigrid (AMG) methods, i.e., the entire multigrid method is built from a linear system,

$$A_1 u_1 = f_1, \tag{1.1}$$

that is provided by the application.

In AMG methods, the time to create the preconditioner can be considerable compared to the time to apply the preconditioner. Within **ML**'s AMG setup, a major computational kernel is the matrix-matrix multiply. It is used in the construction the grid transfer operators that move information to and from coarser levels and in the coarse approximations to the operator $A_1$ in 1.1. Additionally, it is used to form the coarse matrix $A_i, i > 1$, which is often referred to as an *RAP* calculation because $A_i$ is the product of three matrices, $R$, $A_{i-1}$, and $P$. Typically, matrix matrix multiplication accounts for over 50% of the time used to create an **ML** AMG preconditioner.

Applications that have more than one degree of freedom (DOF) per node often lead to block structured matrices. These matrices can be stored in a special format called *variable block row*, in which the DOFs associated with a node are stored in a dense submatrix. This suggests that we may be able to capitalize on the block structure in the setup and execution of the matrix matrix multiply in order to significantly speedup the setup of the AMG preconditioner.

In this paper, we report on a new implementation and initial profiling of a matrix matrix multiply method for variable block matrices. In §2, we give a brief multigrid overview. In §3, we motivate why a block matrix matrix multiply is important to **ML**. In §4, we give an overview of **ML**'s existing point matrix matrix multiply. In §5, we discuss the design and implementation of the block matrix matrix multiply. In §6, we provide some initial numerical profiling results. In §7, we suggest future directions. Finally, in §8 we present the conclusions we draw from our work.

**2. Multigrid Overview.** Multigrid methods (e.g., [6, 8, 1]) are among the most efficient iterative algorithms for solving the linear system, $Ax = f$, associated with elliptic partial

[‡]University of Colorado at Boulder, Department of Computer Science, Ian.Karlin@colorado.edu
[§]Sandia National Laboratories, jhu@sandia.gov

differential equations. The basic idea is to damp errors by utilizing multiple resolutions in the iterative scheme. High-energy (or oscillatory) components are efficiently reduced through a simple smoothing procedure, while the low-energy (or smooth) components are tackled using an auxiliary lower resolution version of the problem (coarse grid). The idea is applied recursively on the next coarser level. An example multigrid iteration is given in Algorithm 1 to solve (1.1). The two operators needed to specify the multigrid method fully are the

**Algorithm 1** Multigrid V-cycle consisting of $N_{levels}$ grids to solve $A_1 u_1 = f_1$.

```
1: {Solve $A_k u_k = f_k$}
2: procedure multilevel($A_k, f_k, u_k, k$)
3: if ($k \neq N_{levels}$) then
4:     $u_k = R_k(A_k, f_k, u_k)$;
5:     $r_k = f_k - A_k u_k$ ;
6:     $A_{k+1} = P_k^T A_k P_k$;
7:     $u_{k+1} = 0$;
8:     multilevel($A_{k+1}, P_k^T r_k, u_{k+1}, k+1$);
9:     $u_k = u_k + P_k u_{k+1}$;
10:    $u_k = R_k(A_k, f_k, u_k)$;
11: else
12:    $u_k = A_k^{-1} f_k$ ;
13: end if
```

relaxation (smoothing) procedures, $R_k$, $k = 1, \ldots, N_{levels}$, and the grid transfers, $P_k$, $k = 2, \ldots, N_{levels}$. Note that $P_k$ is an interpolation operator that transfers grid information from level $k+1$ to level $k$. The coarse grid discretization operator $A_{k+1}$ ($k \geq 1$) is specified by the Galerkin product

$$A_{k+1} = P_k^T A_k P_k. \tag{2.1}$$

The key to fast convergence is the complementary nature of these two operators. That is, errors not reduced by $R_k$ must be well interpolated by $P_k$. While constructing multigrid methods via algebraic concepts presents certain challenges, AMG can be used for several problem classes without requiring a major effort for each application.

**3. Motivation for having a block matrix-matrix multiply.** Applications governed by systems of PDEs often lead to block structured matrices. Examples of such applications are linear elasticity, chemically reacting flow, and compressible flow calculations. These problems have multiple degrees of freedom (DOFs) associated with each grid point (node) in the problem mesh. The group of DOFs at a node comprise a block of coefficients in the matrix. Matrices with block structure can be stored in a variable block row (VBR) structure [2, 9]. The salient feature of this matrix structure is that individual blocks are stored as dense matrices. Hence, accessing column indices require fewer indirect references, and tuned numerical routines may be used for the dense computation.

Profiling of **ML**'s point matrix matrix multiply has shown that the majority of time to calculate the matrix product $AB$ is in the lookup of $B$'s column indices. More specifically, suppose $A$ and (more importantly) $B$ can be stored as VBR matrices. The reduction in lookups of $B$'s column indices is directly related to the block size in $B$. If $B$ has $d \times d$ blocks, then the number of column indices is reduced by a factor of $d^2$, compared to storing $B$ as a point matrix. We note that $d = 3$ is the smallest typical block size. It is not unusual for applications to have $d = 5$ or even larger block sizes. Hence, a reduction of these indirect lookups should lead directly to improvements in the overall runtime.

**4. Overview of the current point matrix matrix multiply.** We first give a high level logical overview of how **ML** performs a matrix-matrix multiply, $A \times B$. For simplicity, $A_i$ denotes the subset of rows of $A$ stored on processor $i$. First, rows of $B$ are exchanged among processors so that processor $i$ has all the information that it needs to calculate $A_i \times B$. Second, the column indices of $B$ are stored in global numbering in a hash table for fast lookup. Third, the local product $A_i \times B$ is calculated. Fourth, the product is converted back to local numbering. Descriptions of the major **ML** functions used in setup and execution of matrix matrix multiplies in **ML** are given in Table 4.1. As mentioned in §1, the matrix-matrix multiply is

TABLE 4.1
*Important functions in* **ML** *for calculating the matrix product $A \times B$.*

| `Function` | `Description` |
|---|---|
| `Convert` | Convert matrix from point to VBR format |
| `Exchange Rows` | Communicates rows of $B$ for the product $A_i \times B$. |
| `Matrix Matrix Multiply` | Performs actual matrix-matrix multiply |
| `Back to Local` | Converts matrix column indices from global to local |
| `Getrow` | Access single point or block row of a block matrix |

an important kernel in the setup of **ML**'s multigrid preconditioners. It is used in the creation of the grid transfer operators, $P_i$, from preliminary transfer operators, $P_i^{(t)}$. For more details on how $P_i^{(t)}$ is constructed, see [10]. Once $P_i^{(t)}$ is available, the prolongator $P_i$ is formed via the step

$$\begin{aligned} P_i &= P_i^{(t)} \\ P_i &\leftarrow (I - \omega_i D_i^{-1} A_i) P_i, \end{aligned} \tag{4.1}$$

where $I$ is an identity matrix, $\omega_i$ is a damping parameter, and $D_i$ is the diagonal of $A_i$. We note that in some cases it is desirable to used repeated applications of (4.1), each of which involves a matrix matrix multiply.

The matrix matrix multiply is also used heavily in the creation of the coarse grid operators $A_i$, $i > 1$. Once $P_i$ and $R_i$ are available, then $A_i$ is formed as in (2.1). Multiplications are performed from right to left. Proceeding in this manner reduces the memory requirements and operation counts in the intermediate product matrices.

**5. Design and Implementation of block matrix matrix multiply.** In this section, we discuss the design and implementation strategy of the block matrix matrix multiply. As mentioned in §3, when the matrix $A$ arises from a system of PDE's, a block matrix multiply has the potential to speedup of the entire multigrid setup, compared to the same calculation with point matrices. This is largely due to multiplication with VBR matrices requiring fewer indirect references.

There are two logical approaches to implementing a block multiplication. In the first approach, every function required to complete the multiplication is refactored to operate natively on block matrices. While this avenue should lead to the best speedups possible, it would also require a large amount of human effort. In the second approach, only certain time-intensive kernels are refactored to operate on block matrices, while the remaining functionality leverages existing point-matrix capabilities.

To keep this project within the scope of a summer, we chose the second approach. Numerical studies in §6 demonstrate that this decision still leads to acceptable overall speedups. In the remainder of this section, we discuss the major phases of the multiplication, our changes to key phases, and potential benefits to refactoring the remaining phases.

The first major component that we implemented is a function that converts point matrices to VBR. This function plays four important roles. First, it was very convenient for testing purposes. It allowed us to use existing point matrices to produce VBR matrices. Second, this method is essential for converting (portions of) a matrix from point to block form after exchange rows has been called. Third, this method allows us to convert an existing $P^{(t)}$ to VBR, rather than having to generate $P^{(t)}$ in VBR format initially. [1] Fourth, this method converts $R$ back to VBR after it is created by transposing $P$. The convert function is sufficiently flexible to be able convert a matrix both before and after rows of that matrix have been communicated. There are two different modes for the function: first, to convert matrices prior to a call to exchange rows; second, to convert the data received by exchange rows. The convert function performs a deep copy of data. An important feature of the convert before exchange rows is called is to ensure that blocks are fully populated (dense) with any missing zeros. By doing so, this speeds up the convert of any exchanged rows.

The second major component that we implemented was two getrow methods. One extracts from a VBR matrix a single point row, and the other extracts a single block row. The capability to extract a point row from a VBR matrix allows us to use any existing **ML** matrix function that requires point row access. In particular, this allowed us to reuse the exchange row function (discussed below). The capability to extract a block row is critical for the core matrix matrix multiply function.

The third major component that we implemented was the matrix matrix multiply kernel. We began this summer project with an existing prototype block multiply. This prototype was capable of squaring a square matrix with $n \times n$ blocks. However, it had several serious limitations. It assumed a fixed block size and worked only in serial. From this prototype, we produced a fully parallel matrix matrix multiply kernel that supports variable block sizes. Tasks included defining a new VBR structure within **ML**, allowing for variable block sizes for the left matrix and a fixed column width for the right matrix, and establishing correct storage estimates for block matrices.

A function that we decided not to refactor is the exchange rows. As mentioned previously in §4, exchange rows must be invoked to communicate rows of $B$ before the product $AB$ can be calculated. Exchange rows accesses matrix data in point fashion (one row at a time). Refactoring this function to access VBR matrices in block fashion could easily have required the entire summer. Moreover, we would have had to ensure that the resulting function's efficiency and scalability were similar to that of the point version. However, because we implemented a VBR matrix getrow that fetches one point row at a time, we were able to reuse the point version of exchange rows.

Refactoring exchange rows may have longer term benefits, however, assuming that a block version has similar performance characteristics to the point version. The cost of data movement of the point version is over 95% of its total cost. A VBR version will still move roughly the same amount of data. However, the data produced by a block exchange row would already be in block format. In contrast, the data from the point version must be converted to block format. The percentage of total time spent in the point exchange row and subsequent

[1]The first phase in which the matrix matrix multiply is used in the creation of $P$ from $P^{(t)}$. (See (4.1).) From initial performance runs it is unclear whether $AP^{(t)}$ multiplication is faster in point or in block form. This is due to the sparsity of the blocks in $P^{(t)}$, which have nonzero entries only on their main block diagonal. If $P^{(t)}$ is in VBR form, all zero entries within a block must be stored explicitly. This increases the effective number of nonzeros by $n^2 - n$ times for relatively small $n \times n$ blocks. This also increases the amount of data that needs to be exchanged in parallel by a corresponding amount in the exchange rows function. Finally, the number of arithmetic operations is increased a factor of $n$, which is not be an important factor in the cost as mentioned in §3 due to the dominant cost of indirect referencing in the matrix-matrix multiply. We estimate that the cost of converting the point matrix $P$ to VBR is 25% of the cost of creating $P$ initially as a VBR matrix.

convert varies with the amount of data on each processor. At 5000 DOFs per processor, the cost is approximately 66% of the total multiply. At 40000 DOFs per processor, the cost is approximately 25%. Regardless of work per processor, we have observed that the conversion from point to block format requires approximately 25% of the time of the exchange row routine. Based on this data, we expect that a block exchange row could decrease the runtime of each multiply by 5-15%. One necessary component that we have not implemented, but that must be, is back to local. In **ML**, the product of two matrices is a matrix with column indices that are globally numbered. In order for the product to be used in subsequent calculations, the column indices must be converted to local numbering. Because the underlying VBR data structure is quite different than that of the **ML** point matrix, **ML** requires a new method to convert VBR matrices from global to local column indices. Without this capability, the conversion to local is possible but is computationally infeasible.

Finally, we decided not to refactor the point matrix transpose. While not a core piece of the matrix matrix multiply, this function is necessary to the calculation (2.1). We expect that the difference in cost between a block and point transpose operation will be similar to that of exchange rows. This is because each is bound by data transfer, and each exchanges approximately the same information between processors. However, the result of the point transpose will be a fairly dense matrix and will therefore be costly to convert. For this reason, we believe that a native block matrix transpose will be beneficial. The effort to write a block transpose should be significantly less than writing a new exchange rows routine.[2]

**6. Results.** Testing and profiling of functions discussed in §5 were performed on the Sandia CSRI machine QED. QED is a 32 node, 64 processor cluster with 2GB of memory per node. Tests were run on three different size matrices, described in Table 6.1. These matrices are typical of those used in elasticity problems and contain $3 \times 3$ subblocks. Processor counts from 1 to 40 were used in tests. The larger matrices were not run on the smallest processor counts due to memory limitations. Each calculation involved squaring the matrix. This was

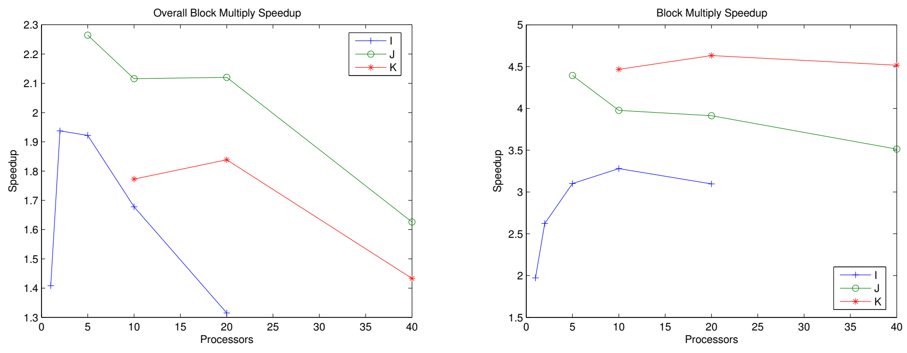
TABLE 6.1
*Test matrices*

| Matrix | Degrees of Freedom | Number of non-zeros |
|---|---|---|
| I | 26460 | 1928958 |
| J | 201720 | 15494286 |
| K | 403440 | 31311086 |

done since it is much easier to setup and run tests in this fashion. These tests should be indicative of the potential performance gains from embedding the block multiply fully within the setup of a multigrid cycle for two reasons. First, in the **ML** RAP process, the intermediate matrices will be have fewer columns than $A$, and therefore require less time to convert and exchange data than with $A$ itself. Second, $3 \times 3$ blocks represent the smallest block size for which the routine can be expected to be used. Other typical sizes such as $3 \times 6$, $5 \times 5$ and $6 \times 6$ will yield larger gains in performance due to less indirect addressing per calculation.

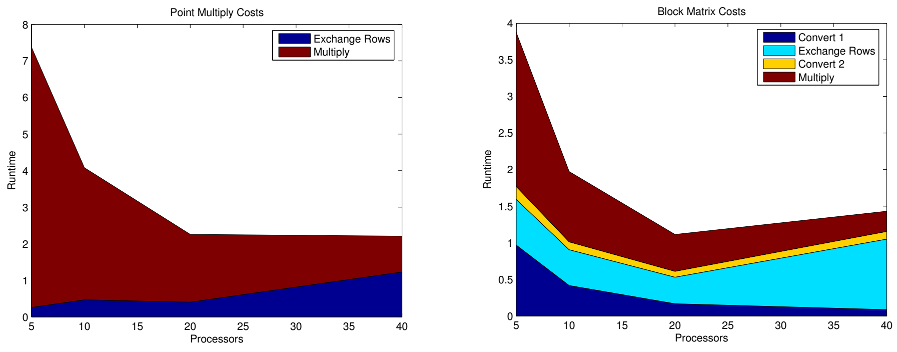
As shown by Figure 6.1(a) the new block multiply results in a 1.3 to 2.3 speedup in the overall multiply calculation. This is due to the 2 to 4.5 speedup of the core multiply routine itself, as shown in Figure 6.1(b). Figures 6.2(a) and 6.2(b) show the component breakdown of the overall costs of the point and block routines for matrix J. The exchange rows function in each routine takes approximately the same time for the same processor count. The main

[2]Note that this must be written from scratch, or after exchange rows is rewritten, as the current transpose routine uses a multiplication by the identity in its operation which requires a call to exchange rows.



(a) Overall Speedup (b) Multiply Speedup

FIG. 6.1. *Performance gains from block multiply*



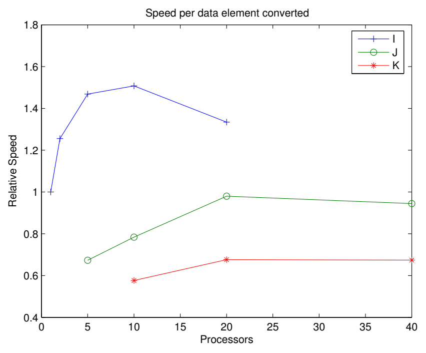
(a) Point Multiply (b) Block Multiply

FIG. 6.2. *Component costs for multiplying the J matrix*

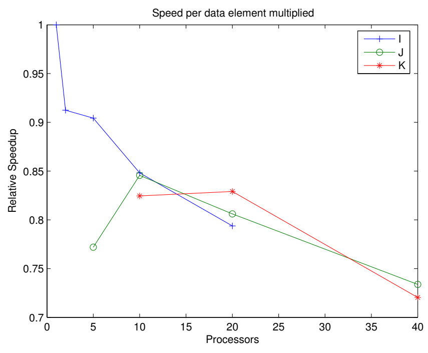
advantage of the block routine is from the reduced cost of the multiply routine. The time spent in the two conversions, however, offsets some of this reduction. The conversions account for approximately 25% of the overall runtime, and are a potential spot for further optimization.

Note that the time for exchange rows in both routines increases when moving from 20 to 40 processors. As there was no attempt to load balance other than the equal distribution of rows among processors this 3 fold increase could be due to a bad data exchange pattern or a bad parallel distribution of matrix rows. In a real application load balancing would likely fix this issue. Figures 6.3(a) and 6.3(b) show the scaling of the convert of the B matrix to VBR and the block multiply. The results are normalized to the speed per nonzero of the I matrix running in serial. Scaling of exchange rows is not shown as previous work has explored its scaling properties, and no work was done on this function during this project. The scalability of the second convert was not studied as its cost is approximately 25% of cost of exchange rows.

What is shown in 6.3(a) is the convert becomes more efficient per nonzero converted as the work per processor decreases up to a certain point, where the trend reverses. In addition for larger matrices the convert is less efficient than for smaller ones. For the multiply 6.3(b) shows that the scaling of the multiply is tied to the number of processors used for the problem. With the exception of the 5 processor example for the J matrix, the efficiency of



(a) Convert Scaling



(b) Multiply Scaling

FIG. 6.3. *Relative speed of convert and multiply routines*

the computation is nearly identical for each matrix when the number of processors is held constant.

**7. Future work.** To fully integrate the block matrix-matrix multiply into the **ML** multigrid setup phase, a few functions need to be finished. More specifics are outlined in the ML developers documentation [5]. A VBR version of back to local should be written. The writing of a wrapper routine modeled after the current driver and `ML_2matmult()` would make the routine much more accessible for a developer to call.

Within the current design approach, if one were looking for additional efficiency, the following are the best candidates for performance gains. By changing the convert to handle matrices exchanged in point format, $P^{(t)}$ could be more efficiently exchanged. This may increase the convert time on the exchanged rows but would decrease the exchanged information to $1/n$ of its current amount, where $n \times n$ is the block size. The convert routine has not been profiled, and there is a chance it has inefficiencies that could eliminated. Also, while the multiply has no obvious inefficiencies, it may benefit from calls to BLAS[3] routines, especially for larger block sizes. Profiling of this routine might uncover other areas for improvements, though this is unlikely as it was derived from an efficient point multiply. A VBR matrix vector multiply could also lead to performance gains in the application of the multigrid preconditioner.

If full fledged VBR support were desired, we suggest the following order for the implementation. First if a VBR transpose is easy to write, or if an EPETRA function can be utilized, this would be the easiest function to write with potentially the largest performance gains. If the transpose is not easy or requires a block exchange row function to work, then the creation of a block $P^{(t)}$should be the first priority. A new exchange rows function should be lowest priority, unless a block transpose requires it. This is because the expected reduction in runtime of a new block exchange rows is small in comparison to the effort to refactor the code.

**8. Conclusions.** This report summarizes a summer project to implement a block matrix-matrix multiply within the multigrid preconditioning package **ML**. We have demonstrated 2-4.5 times speedups in the multiply kernel for linear systems with $3 \times 3$ blocks, and overall speedups of 1.3-2.3, although these results are likely a lower bound on actual performance. Development time was dramatically reduced through the use of a point-to-VBR converter function and existing point matrix capabilities, while still allowing for significant speedups. While we chose to refactor only portions of the multiply, we believe that the results from the initial profiling show this decision was correct.

## REFERENCES

[1] W. L. Briggs, V. E. Henson, and S. McCormick, *A multigrid tutorial, Second Edition*, SIAM, Philadelphia, 2000.

[2] S. Carney, M. Heroux, and G. Li, *A proposal for a sparse BLAS toolkit*, tech. report, Cray Research Inc., Eagen, MN, 1993.

[3] J. J. Dongarra, J. D. Croz, S. Hammarling, and I. Duff, *A set of level 3 Basic Linear Algebra Subprograms*, ACM Transactions on Mathematical Software, 16 (1990), pp. 1–17.

[4] M. W. Gee, C. M. Siefert, J. J. Hu, R. S. Tuminaro, and M. G. Sala, *ML 5.0 smoothed aggregation user's guide*, Tech. Report SAND2006-2649, Sandia National Laboratories, 2006.

[5] M. W. Gee, C. M. Siefert, J. J. Hu, R. S. Tuminaro, M. G. Sala, and I. Karlin, *ML developer's guide*. 2007.

[6] W. Hackbusch, *Multigrid Methods and Applications*, vol. 4 of Computational Mathematics, Springer–Verlag, Berlin, 1985.

[7] M. A. Heroux, R. A. Bartlett, V. E. Howle, R. J. Hoekstra, J. J. Hu, T. G. Kolda, R. B. Lehoucq, K. R. Long, R. P. Pawlowski, E. T. Phipps, A. G. Salinger, H. K. Thornquist, R. S. Tuminaro, J. M. Willenbring, A. Williams, and K. S. Stanley, *An overview of the Trilinos project*, ACM Trans. Math. Softw., 31 (2005), pp. 397–423.

[8] U. Trottenberg, C. Oosterlee, and A. Schüller, *Multigrid*, Academic Press, London, 2001.

[9] R. S. Tuminaro, M. Heroux, S. A. Hutchinson, and J. N. Shadid, *Official Aztec user's guide version 2.1*, Tech. Report SAND99-8801J, Sandia National Laboratories, 1999.

[10] P. Vaněk, J. Mandel, and M. Brezina, *Algebraic multigrid based on smoothed aggregation for second and fourth order problems*, Computing, 56 (1996), pp. 179–196.

# MODEL REDUCTION BY COMPONENT MODE SYNTHESIS: CRAIG-BAMPTON METHOD WITH LINEAR EXAMPLE

REGINA M. DAVIS[‡] AND RICHARD B. LEHOUCQ[§]

**Abstract.** Component mode synthesis (CMS) is a model reduction technique in which the components of a larger structure are condensed to reduced-order models and reassembled to produce a reduced-order model of the entire system. In this document, a brief introduction of CMS will be given and then Craig-Bampton's method for CMS will be explained in more detail and used to model the linear response of a rod, spring at one end and free-ended at the other, when a ping excitation occurs at its free end. The intent of this document is an elementary set of notes based on a first time reading of CMS methods.

**1. Introduction.** In structural dynamics, the topic of model reduction surfaces upon consideration of large multi-degree of freedom systems. Due to the complexity of such structures, solutions to the large system of equations become burdensome and costly. Finite element models, (although widely used, highly favored, and perhaps the most powerful models used for complex systems), are unfortunately expensive in these cases as they are difficult to pull together, and untimely to produce. Model reduction is then the theory of approximating higher order systems by lower order systems while preserving as much of the systems' behavior as possible in an effort to increase the time efficiency and decrease the expense of the analysis.

Component mode synthesis employs the idea of dividing a large structure into smaller sub-structures, investing most of the analysis into these smaller components, and in turn reaping a decent approximation of the complete system. Component mode synthesis has several applications, primarily in coupling reduced-order models of components in a larger system, test verification of finite element models, and to achieve an understanding of the dynamics of multimillion-DOF models [1, ch.17 pp.532].

The underlying concept behind CMS is the notion that the components' physical coordinates can be represented in terms of a set of generalized coordinates from the following transformation:

$$\mathbf{q} = \mathbf{C}\mathbf{p}. \tag{1.1}$$

The transformation matrix, $\mathbf{C}$, also called the *component mode matrix*, pre-multiplies the generalized coordinates, $\mathbf{p}$, that describe the motion of the system. The columns of the component mode matrix, called *component modes*, are assumed modes describing the physical displacements, $\mathbf{q}$, of each coordinate.

There are several methods in CMS which can be used, depending on the components' geometries and boundary conditions. While each method contributes to the collective strength of CMS, discussion will be limited to the Craig-Bampton fixed-interface method. The intent of this document is an elementary set of notes based on a first time reading of CMS methods. The reader is referred to [1, ch.17 pp.532] and the open literature for surveys and more involved discussions.

**2. Basics.** Consider a clamped-free rod divided as shown in Figure 2.1(a).

The structure has been divided into three components, $\alpha$, $\beta$, and $\gamma$. The $\alpha$ component is broken down into internal coordinates and interface coordinates (Figure 2.1(b)), where $I$ represents the internal coordinates and $E$ represents the interface coordinates[1, ch.17 pp.533]. An *internal coordinate* is any coordinate within the component not associated with any

[‡]New Mexico State University, reginab@nmsu.edu

[§]Sandia National Laboratories, rblehou@sandia.gov

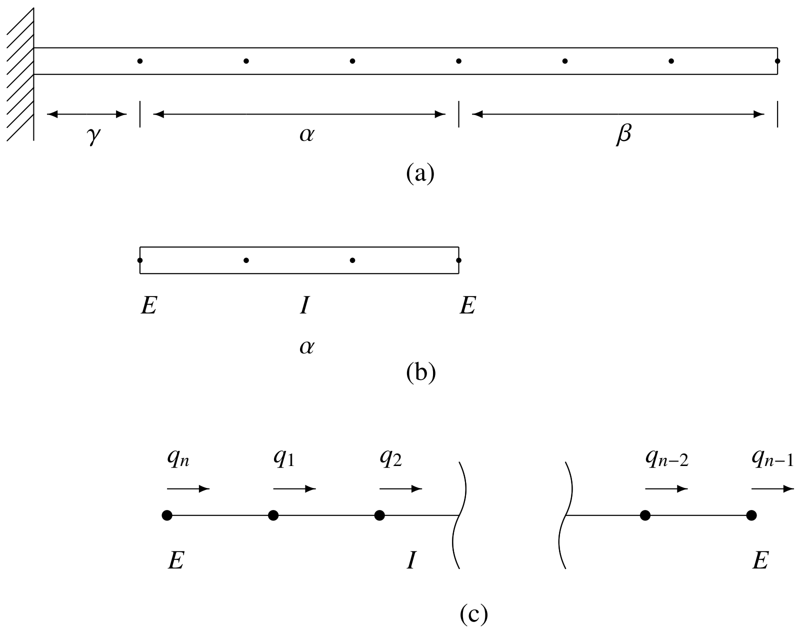


FIG. 2.1. *(a)Clamped-free rod divided into $\alpha$, $\beta$, and $\gamma$ components (b)the $\alpha$ component with internal and interface coordinates shown (c)schematic of a typical component with n-DOF*

boundary or constraint. An *interface coordinate* is any coordinate within the component for which another component is coupled. Interface coordinates between components are related by a set of constraints. The $\alpha$ component, as shown, has two internal coordinates and two interface coordinates.

The component mode matrix for the $\alpha$ component is composed of two types of assumed modes: constraint modes and fixed interface modes. A *constraint mode* is a static solution where all but one boundary[1] DOF is held zero, and the remaining one is set to one. A *fixed-interface mode* is a static solution where all boundary DOF are held zero. Each system has a set of fixed-interface modes equal to the number of interior nodes.

Similarly, the component mode matrix for the $\beta$ component is also composed of constraint modes and fixed interface modes. Although the right boundary for the $\beta$ component is not an interface coordinate, the required constraints on the component allow for it to be modeled as one.

Constraint modes and fixed-interface modes are the two component modes used in Craig-Bampton's method. There are other component modes included in CMS, such as rigid-body modes as would appear in the component mode matrix for the $\gamma$ component due to the the rigid-body coordinate on the left boundary, but these are not needed in Craig-Bampton's method nor will they appear in the example to follow. They will therefore not be discussed.

One final feature of the component, which may be obvious but will be presented anyhow, is that it can be modeled with as many elements as desired. While only three elements of

[1]in these definitions, a component's *boundary* is simply any coordinate lying on the geometric boundary of the component; in the case of the $\alpha$ component, the boundaries are represented by the interface coordinates.

the $\alpha$ component have been illustrated, it could potentially have $n$-DOF with the schematic shown in Figure 2.1(c).

**3. The Component Mode Matrix by Craig-Bampton Method.** The component mode matrix will be developed from Craig-Bampton's fixed-interface transformation. The discussion that follows and discussion in Section 4 resemble that in Craig and Kurdila[1, ch.17 pp.532-37,557-59].

For simplification and for clarity, the interface coordinates will be referred to as the boundaries. Thus, $E$ will be called $B$ and the order $I \to B$ will always be used. It follows, the stiffness matrix, $\mathbf{K}$, for the component in Figure 2.1(c) has the form[2]

$$\mathbf{K} = \begin{bmatrix} \mathbf{K}_{ii} & \mathbf{K}_{ib} \\ \mathbf{K}_{bi} & \mathbf{K}_{bb} \end{bmatrix}.$$

Likewise, the mass matrix, $\mathbf{M}$, is

$$\mathbf{M} = \begin{bmatrix} \mathbf{M}_{ii} & \mathbf{M}_{ib} \\ \mathbf{M}_{bi} & \mathbf{M}_{bb} \end{bmatrix}.$$

The component mode matrix, $\mathbf{C}$, will consist of constraint and fixed-interface modes as described in Section 2. The physical coordinates can then be represented as a linear combination of the constraint modes and the fixed-interface modes as follows:

$$\mathbf{q} = \Phi_k \mathbf{p}_k + \Psi_c \mathbf{p}_c. \tag{3.1}$$

Where $\Phi_k$ and $\Psi_c$ represent the fixed-interface modes and the constraint modes respectively. The subscript $k$ is used to denote the number of retained, or kept, fixed-interface modes and $\mathbf{p}_k$ refers to the generalized coordinates corresponding to the fixed-interface modes. The subscript $c$ is used to denote the number of constraint modes and likewise, $\mathbf{p}_c$ refers to the generalized coordinates corresponding to the constraint modes.

The modes can be divided into internal ($i$) and boundary ($b$) coordinates,

$$\Phi_k = \begin{pmatrix} \Phi_{ik} \\ \Phi_{bk} \end{pmatrix}, \quad \Psi_c = \begin{pmatrix} \Psi_{ic} \\ \Psi_{bc} \end{pmatrix}.$$

By their definition,

$$\begin{aligned} \Phi_{bk} &= \mathbf{0}_{bk}, \\ \Psi_{bc} &= \mathbf{I}_{bc}. \end{aligned}$$

Thus,

$$\Phi_k = \begin{pmatrix} \Phi_{ik} \\ \mathbf{O}_{bk} \end{pmatrix}, \quad \Psi_c = \begin{pmatrix} \Psi_{ic} \\ \mathbf{I}_{bc} \end{pmatrix}. \tag{3.2}$$

The number of constraint modes is equal to the number of boundaries on the component, ($c = b$). Replacing $c$ with $b$ in (3.2) and expanding ( 3.1) gives

$$\mathbf{q} = \begin{pmatrix} \mathbf{q}_i \\ \mathbf{q}_b \end{pmatrix} = \begin{bmatrix} \Phi_{ik} & \Psi_{ic} \\ \mathbf{0}_{bk} & \mathbf{I}_{bb} \end{bmatrix} \begin{pmatrix} \mathbf{p}_k \\ \mathbf{p}_b \end{pmatrix}, \tag{3.3}$$

[2]If $r$ and $c$ are the number of rows columns of a matrix respectively, then any matrix having the notation $\mathbf{A}_{rc}$ has the condition $\mathbf{A}_{rc} \in \mathbb{R}^{r \times c}$; and any vector $\mathbf{a}_r$ is a column array with dimensions $(r \times 1)$. Also note some matrices are only given one subscript. This does not denote a dimension; it only gives the matrix a label (see eqn 3.1).

for which the component mode matrix is

$$\mathbf{C} = \begin{bmatrix} \Phi_{ik} & \Psi_{ib} \\ \mathbf{0}_{bk} & \mathbf{I}_{bb} \end{bmatrix}.$$

The internal partition of the fixed-interface modes, $\Phi_{ik}$, is determined from the following eigenproblem:

$$\left(\mathbf{K}_{ii} - \left(\omega_i^2\right)_k \mathbf{M}_{ii}\right)(\phi_i)_k = \mathbf{0}$$

or

$$(\mathbf{K}_{ii} - \mathbf{M}_{ii}\Omega_{ii})\,\Phi_{ii} = \mathbf{0}.$$

where $\Phi_{ik}$ is composed of the first $k$ modes of $\Phi_{ii}$. The internal partition of the constraint modes, $\Psi_{ib}$, however, are determined from the following static definition:

$$\begin{bmatrix} \mathbf{K}_{ii} & \mathbf{K}_{ib} \\ \mathbf{K}_{bi} & \mathbf{K}_{bb} \end{bmatrix} \begin{pmatrix} \Psi_{ib} \\ \mathbf{I}_{bb} \end{pmatrix} = \begin{pmatrix} \mathbf{0}_{ib} \\ \Gamma_{bb} \end{pmatrix},$$

for which $\Gamma_{bb}$ are the reaction forces at each constraint. Solving:

$$\Psi_{ib} = -\mathbf{K}_{ii}^{-1}\mathbf{K}_{ib}$$

and

$$\Psi_c = \begin{pmatrix} -\mathbf{K}_{ii}^{-1}\mathbf{K}_{ib} \\ \mathbf{I}_{bc} \end{pmatrix}.$$

Recall, $n$ is the DOF, $n = i + b$, and $k$ is the number of kept fixed-interface modes. For clarity, observe the following dimensions for (3.3)

$$\begin{matrix} \mathbf{q} \\ (n \times 1) \end{matrix} = \begin{pmatrix} \mathbf{q}_i \\ (i \times 1) \\ \\ \mathbf{q}_b \\ (b \times 1) \end{pmatrix} = \begin{bmatrix} \Phi_{ik} & \Psi_{ib} \\ (i \times k) & (i \times b) \\ & \\ \mathbf{0} & \mathbf{I} \\ (b \times k) & (b \times b) \end{bmatrix} \begin{pmatrix} \mathbf{p}_k \\ (k \times 1) \\ \\ \mathbf{p}_b \\ (b \times 1) \end{pmatrix}$$

**4. Reducing the Model.** Model reduction in CMS occurs due to the transformation matrix, **C**, in (1.1), by which the physical coordinates, **q**, are described by a reduced number of generalized coordinates, **p**. Let $m$ be the number of generalized coordinates, where $m < n$. Equation 1.1 can be expressed with dimensions as

$$\underbrace{\mathbf{q}}_{n\times 1} = \underbrace{\mathbf{C}}_{n\times m}\underbrace{\mathbf{p}}_{m\times 1}. \tag{4.1}$$

The equation of motion for the undamped system is:

$$\mathbf{M}\ddot{\mathbf{q}} + \mathbf{K}\mathbf{q} = \mathbf{f}. \tag{4.2}$$

Substitution of (4.1) into (4.2) gives

$$\mathbf{M}\mathbf{C}\ddot{\mathbf{p}} + \mathbf{K}\mathbf{C}\mathbf{p} = \mathbf{f}.$$

To ensure the orthogonality of all the component modes to the residuals, apply Galerkin's method. Require

$$\mathbf{y}^T\,(\,\mathbf{f} - \mathbf{MC}\ddot{\mathbf{p}} + \mathbf{KCp}) = \mathbf{0}, \qquad \forall\;\mathbf{y} \in span\,\{\mathbf{C}\}\,. \tag{4.3}$$

Therefore,

$$\mathbf{C}^T\mathbf{MC}\ddot{\mathbf{p}} + \mathbf{C}^T\mathbf{KCp} = \mathbf{C}^T\mathbf{f}$$

or

$$\mathbf{M}_r\ddot{\mathbf{p}} + \mathbf{K}_r\mathbf{p} = \mathbf{f}_r,$$

where

$$\mathbf{M}_r = \mathbf{C}^T\mathbf{MC}, \quad \mathbf{K}_r = \mathbf{C}^T\mathbf{KC}, \quad \mathbf{f}_r = \mathbf{C}^T\mathbf{f}.$$

Note, Galerkin's condition in (4.3) is also the principle of virtual work.

**5. Understanding the Reduction.** Now the question may arise, "How does this reduction work?" Or another to ask is "What effect will this reduction have on the solution to the system's dynamic behavior?" In an attempt to answer these questions, the reduction will be broken down into finer detail by way of the the homogeneous solution to the equation of motion,

$$\mathbf{M}\ddot{\mathbf{x}} + \mathbf{Kx} = \mathbf{0}. \tag{5.1}$$

Assume an exponential solution,

$$\mathbf{q} = \mathbf{x}e^{\omega t}. \tag{5.2}$$

Consider substitution of (5.2) into (5.1). The result yields the eigenvalue problem

$$\left(\mathbf{K} - \left(\omega_o^2\right)_j \mathbf{M}\right)(\mathbf{x}_o)_j = \mathbf{0} \qquad j = 1, 2, \cdots, n$$

or

$$(\mathbf{K} - \mathbf{M}\Omega_o)\,\mathbf{X}_o = \mathbf{0}, \tag{5.3}$$

where the subscript $o$ denotes the eigenvectors, modes, and eigenvalues, frequencies, of the un-reduced $n$-DOF system. Now, let

$$\mathbf{p} = \mathbf{y}e^{\omega t}. \tag{5.4}$$

Substitution of (5.4) into (1.1) gives

$$\mathbf{q} = \mathbf{Cy}e^{\omega t}. \tag{5.5}$$

Given $\mathbf{C}$ is square, $(n \times n)$, the resulting eigenvalue problem is

$$\left(\mathbf{C}^T\mathbf{KC} - \left(\omega_o^2\right)_j \mathbf{C}^T\mathbf{MC}\right)(\mathbf{y}_o)_j = \mathbf{0} \qquad j = 1, 2, \cdots, n$$

or

$$(\mathbf{K}_r + \mathbf{M}_r\Omega_o)\,\mathbf{Y}_o = \mathbf{0}, \tag{5.6}$$

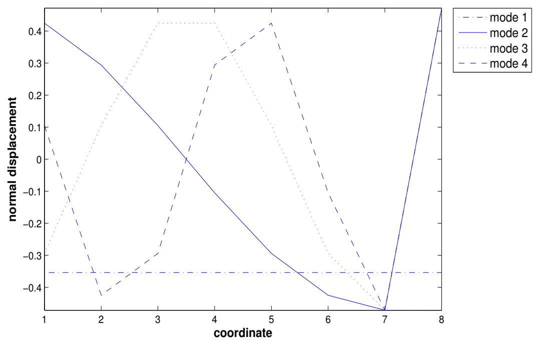


FIG. 5.1. *The first four modes of the original, 8 DOF system (in blue) are compared to the first four modes in the unreduced, 8 DOF CMS problem (in red). The CMS modes cannot be seen because they are identical to the true modes of the original system.*

and by (5.2) and (5.5),

$$\mathbf{X}_o = \mathbf{C}\mathbf{Y}_o. \tag{5.7}$$

Again, the subscript $o$ denotes the un-reduced system. Here, $\mathbf{Y}_o$ represents the transformed eigenvectors of the un-reduced system. Equation 5.6 suggests that, for any $(n \times n)$ $\mathbf{C}$, the resulting modes, $\mathbf{X}_o$, and frequencies, $\Omega_o$ are identical to those in (5.3).

However, when $m < n$, the resulting modes and frequencies are similar to those of the un-reduced system. In other words, for

$$(\mathbf{K}_r + \mathbf{M}_r\Omega)\,\mathbf{Y} = \mathbf{0},$$

then

$$\mathbf{X} \approx \mathbf{X}_o \tag{5.8}$$

and

$$\Omega \approx \Omega_o. \tag{5.9}$$

The relationship in (5.7) is no longer true, it is only an approximation. The relationship between $\mathbf{X}$ and $\mathbf{Y}$ can be determined from Galerkin's condition in a least squares sense. This results in[3]

$$\mathbf{X} \approx \mathbf{C}\mathbf{Y} = \mathbf{C}\left[\mathbf{C}^T\mathbf{C}\right]^{-1}\mathbf{C}^T\mathbf{X},$$

or

$$\mathbf{Y} = \left[\mathbf{C}^T\mathbf{C}\right]^{-1}\mathbf{C}^T\mathbf{X}. \tag{5.10}$$

Equations 5.8 and 5.9 give rise to the following:

$$\begin{aligned}(\mathbf{K} + \mathbf{M}\Omega_o)\,\mathbf{X}_o &= \mathbf{0},\\ (\mathbf{K} + \mathbf{M}\Omega)\,\mathbf{X} &= \mathbf{r}.\end{aligned}$$

[3]The columns of $\mathbf{C}$ are linearly independent and therefore the inverse of $\mathbf{C}^T\mathbf{C}$ exists.

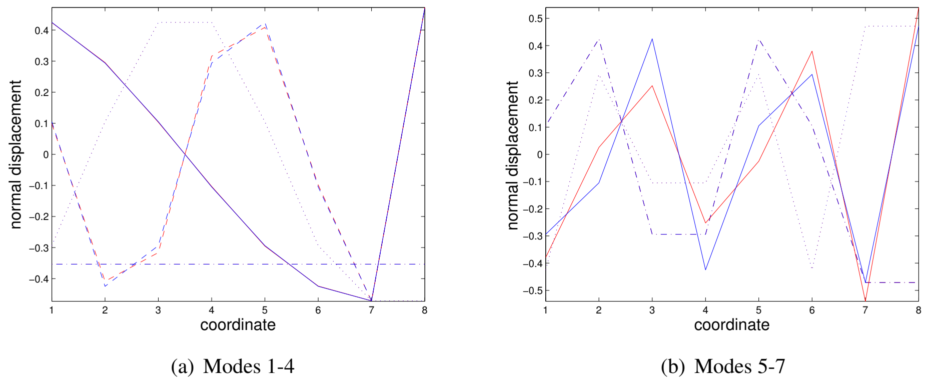


FIG. 5.2. *(a)The first four modes of the original, 8 DOF system (in blue) are compared to the first four modes of the reduced, 7 DOF CMS problem (in red). (b)The remaining three modes of the reduced, 7 DOF CMS problem are compared to modes 5-7 of the original unreduced system.*

In an effort to compare the residuals for each reduction, consider:

$$\|\mathbf{r}\|_{\mathbf{k}^{-1}} = \sqrt{\mathbf{r}^T\mathbf{K}^{-1}\mathbf{r}} \tag{5.11}$$

For the case when $m = n$,

$$\|\mathbf{r}_o\| = 0$$

In all other cases where $m < n$,

$$\|\mathbf{r}\| > 0$$

Let the residual vector for each reduction, $\mathbf{r}$, be an $(m \times 1)$ array of the residuals for each mode.

$$\mathbf{r} = \begin{bmatrix} \|\mathbf{r}_1\| & \|\mathbf{r}_2\| & \cdots & \|\mathbf{r}_m\| \end{bmatrix}^T$$

**6. Linear Homogeneous Solutions.** Perhaps the most straightforward representation of how the reduced model compares to the original, full scale model, is the comparison of the homogeneous solution each model predicts. Take, the linear, free-free ended, homogeneous $n$-DOF system shown in the schematic in Figure 6.1 with the following conditions:

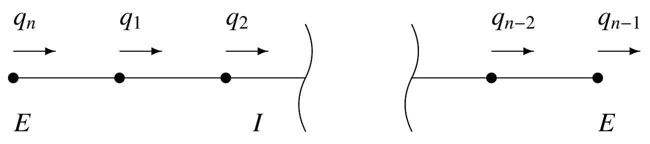


FIG. 6.1. *Free-free ended component with n-DOF, no bending*

$$\mathbf{M}\ddot{\mathbf{q}} + \mathbf{K}\mathbf{q} = \mathbf{0},$$
$$\mathbf{q}(0) = \mathbf{x}_j,$$

where

$$\left(\mathbf{K} - \omega_j \mathbf{M}\right)\mathbf{x}_j = \mathbf{0} \qquad j = 1, 2, \cdots, N_n$$

The solution to this system is simply,

$$\mathbf{q} = \mathbf{x}_j \cos(\omega_j t) \tag{6.1}$$

Replacing $\mathbf{q}$ with $\mathbf{Cp}$ in this example gives, for $\mathbf{C} \in \mathbb{R}^{n\times m}$ and $m < n$,

$$\mathbf{M}_r\ddot{\mathbf{p}} + \mathbf{K}_r\mathbf{p} = \mathbf{0},$$
$$\mathbf{p}(0) = \mathbf{y}_j,$$

where

$$\left(\mathbf{K} - \nu_j \mathbf{M}\right)\mathbf{y}_j = \mathbf{0} \qquad j = 1, 2, \cdots, N_m$$

Similarly, the solution to this system is,

$$\mathbf{p} = \mathbf{y}_j \cos(\nu_j t). \tag{6.2}$$

The solution for the generalized coordinates in (6.2) can be compared to the "true" solution for the physical coordinates in (6.1) by the transformation matrix $\mathbf{C}$:

$$\tilde{\mathbf{q}} = \mathbf{C}\mathbf{y}_j \cos(\nu_j t). \tag{6.3}$$

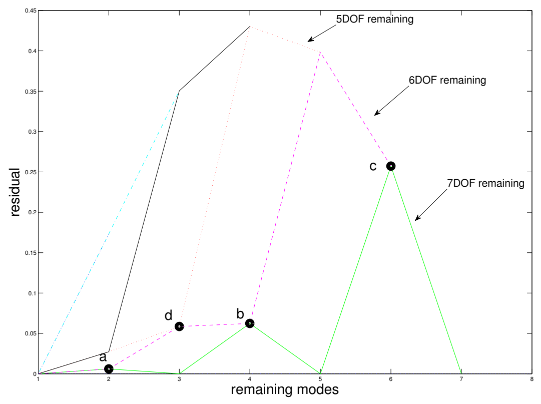
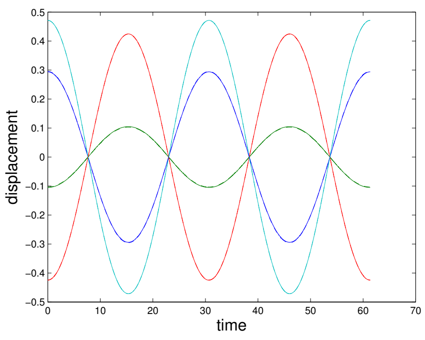
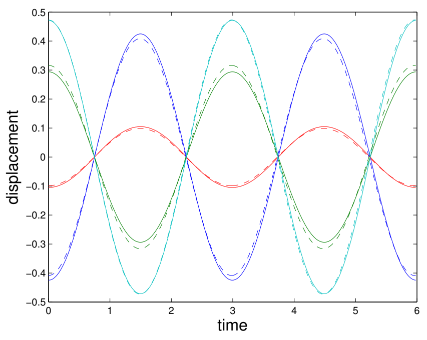


Fig. 6.2. *Residuals for the 8DOF with 1, 2, ..,6 reductions; (a)one reduction, seven modes remaining, second mode - Fig 6.3(a); (b)one reduction, seven modes remaining, fourth mode - Fig 6.3(b); (c)one reduction, seven modes remaining, sixth mode - Fig 6.3(c); (d)three reductions, five modes remaining, third mode - Fig 6.3(d)*

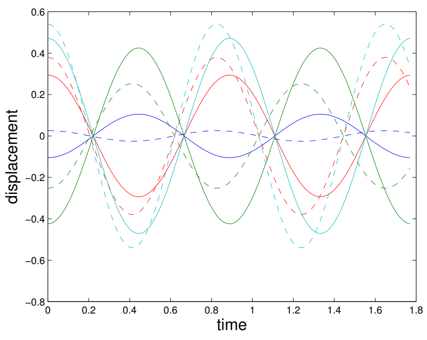
An 8DOF model with the geometry shown in Figure 6.1, has two constraint modes and six fixed-interface modes. In reduction, the constraint modes and low-frequency fixed-interface modes are retained. Figure 6.2 shows the residual progression as fixed-interface modes are removed one at a time. Figure 6.3 shows the corresponding displacements for four of the coordinates, $\mathbf{q}$ (6.1), compared to the corresponding CMS approximations, $\tilde{\mathbf{q}}$ (6.3), labeled in Figure 6.2. Note the value of the residual suggests how much the approximated CMS solution has strayed from the true solution.



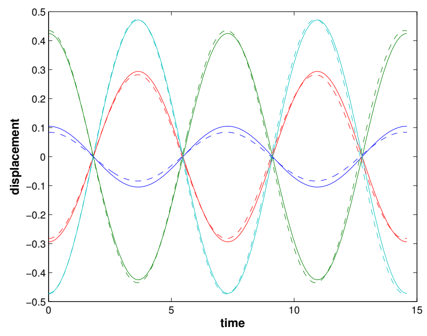
(a) Second mode oscillation after one reduction



(b) Fourth mode oscillation after one reduction



(c) Sixth mode oscillation after one reduction



(d) Third oscillation after three reductions

Fig. 6.3. *(a)The displacements for four coordinates in the fourth mode for the full, 8DOF model. (b)The displacements for four coordinates in the fourth mode after one reduction is made. Solid curves are the"true" displacements and the dashed curves are displacements obtained from one reduction.*

For simplicity, the above illustration was made by way of an 8DOF, free-free ended, homogeneous system. Higher-ordered systems provide for better approximations by CMS. Consider the residual plot for the 16DOF in Figure 6. The above process of comparing the residuals to the true and approximated displacements was repeated for the 16DOF, 32DOF, and 64DOF system and the following observations have been made: situations which lie on the outside line of the residual plot have experienced a significant change in frequency and are noticeably poor approximations to the true solution due to their large change in frequency, as observed the CMS approximations in Figure 6.3(c) with 6.2; all other situations lying on the $n-2$ line or below are relatively good approximations to the true solution, (Figure 6.3(a,b,d) with 6.2).

**7. A Linear Application of CMS.** In the previous example, the routines were formulated so as to produce only one mode and frequency in the displacement solutions. Problems of more interest contain a combination of all the modes in their solution. CMS will now be applied to a practical example in order to gain a better understanding of the concept of a reduced model and CMS capabilities.

Consider the spring-free rod in Figure 7.1. Let initial displacements and velocities for all coordinates be equal to zero, and let the impulse be applied at $t = 0$.

As an example, Figure 7 shows the accelerations at the right end of a 32DOF due to a short impulse at time $t = 0$ (cyan). The plot also shows the CMS acceleration approximations

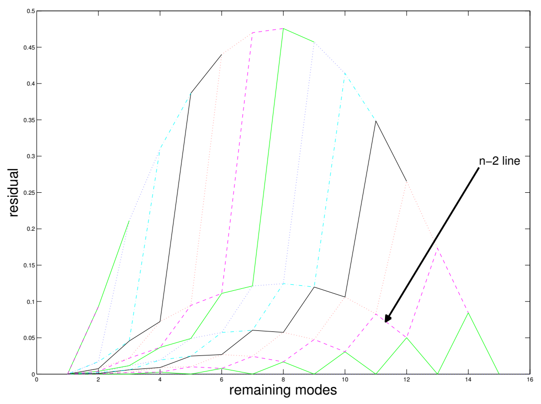


FIG. 6.4. *Residuals for the 16DOF, free-free ended, homogeneous systems*

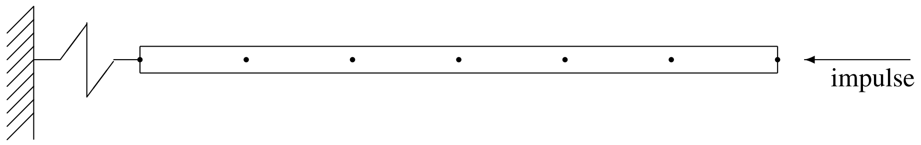


FIG. 7.1. *Rod with linear spring on one end and free-ended at the other end. Short impulse applied at time t=0.*

when only 8 component modes are used to estimate the same response (red). Note the absence of higher frequency oscillation in the reduced-order model. This is due to the fact that the 8 modes retained in the reduction were those corresponding to the eight lowest frequencies from the eigenvalue problem described in Section 5. In other words, the high frequencies that appear in the real solution have been removed, or discarded to obtain the reduced solution.

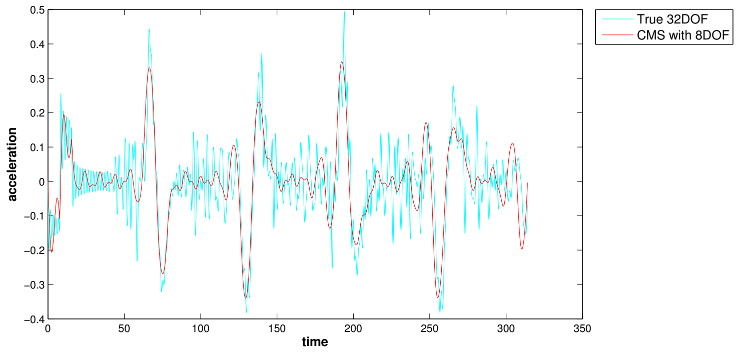


FIG. 7.2. *True acceleration response at the rod's free end for 32DOF when short impulse is applied at t=0 (cyan) compared to CMS approximation for the same response modeled as 8DOF (red).*

**8. Summary.** In short, Component Mode Synthesis and its applications in complex structures has been explained. Craig-Bampton's method of using constraint modes and fixed-interface modes to form the component mode matrix has also been explained and was used to demonstrate CMS approximations in linear examples. Homogeneous displacement solutions were used to emphasize the details of how CMS approximations compare to true solutions; and finally CMS was applied to a practical linear example in order to illustrate the transformation that takes place under CMS. Figure 7 suggests, thus far, that CMS is an appropriate model reduction method under the linear conditions described in this document. Further investigation under additional conditions such as bending, non-linearities, 2-D models, multiple components are certainly other areas of interest in CMS, but are not discussed in this document.

## REFERENCES

[1] R. R. Craig, Jr. and A. J. Kurdila, *Fundamentals of Structural Dynamics*, John Wiley and Sons, Inc., second ed., 2006.

# CONSTRAINED EIGENVALUE PROBLEMS

CHRISTOPHER G. BAKER[§] AND RICHARD B. LEHOUCQ[¶]

**Abstract.** This note proposes an improved algorithm for the numerical solution of symmetric eigenvalue problems with constraints in Salinas. We briefly review the current approach, explain its deficiencies, and then propose a new algorithm. In addition to the algorithm's improved stability, an inner iteration is not needed—an application of the preconditioner is all that is required. Moreover, redundant constraints (characterized by a rank deficient constraint matrix) are not problematic.

**1. Introduction.** The numerical solution of the constrained eigenvalue problem

$$\begin{bmatrix} K & C^T \\ C & 0 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} M & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \lambda \,. \tag{1.1}$$

for the low energy modes is required by Salinas [2]. The matrices $K, M \in \mathbb{R}^{n\times n}$ are symmetric/positive semi-definite. The matrix $C$ is $m \times n$, $r = \operatorname{rank}(C)$, $r \leq m < n$, and is not assumed to be of full row rank.

The eigenvalue problem (1.1) is the optimality system for the following constrained eigenvalue problem: Find $(x, \lambda)$ so that

$$Kx = Mx\lambda \quad \text{subject to} \quad Cx = 0. \tag{1.2}$$

The vector $y$ represents the Lagrange multipliers.

**2. Existing approach.** The existing approach in Salinas is to directly attack the augmented eigenvalue problem (1.1). This is done by using the ARPACK/PARPACK software in a shift-invert mode [5, 6]. The shift-invert mode requires solving for $w$ systems of the form

$$\left( \begin{bmatrix} K & C^T \\ C & 0 \end{bmatrix} - \sigma \begin{bmatrix} M & 0 \\ 0 & 0 \end{bmatrix} \right) w = \begin{bmatrix} M & 0 \\ 0 & 0 \end{bmatrix} z \,. \tag{2.1}$$

The reverse-communication interface of ARPACK requires only that Salinas provide methods for applying the Krylov subspace operator, the shift-invert operator, and the augmented mass matrix. Unfortunately, this approach is unstable and stems from using a preconditioned inner iteration to approximate $w$ even if the associated residual is small. Because the augmented system (1.1) has infinite eigenvalues, small errors in the approximation to $w$ computed can be dramatically amplified. Moreover, rank deficiency in $C$ compounds this amplification because then the linear system (2.1) does not have a unique solution. Expecting the Krylov based eigensolver in Anasazi to compute the same eigenpairs as ARPACK is a foolhardy task.

One drawback of a shift-invert approach such as above is the need to accurately solve the linear system. By exploiting one of a member of the class of preconditioned iterative eigensolvers (see, for example, [1]), this requirement on the exact solve can be relaxed for use as a preconditioner. The benefit of this is that the accuracy of the linear solve affects only the convergence of the eigensolver, not its solutions. This should be contrasted with a shift-invert Krylov subspace method, where the solutions of eigensolver are defined by the accuracy of the shift-invert solver.

The preconditioned iterative solvers, such as the Block Davidson solver [1] and the Locally Optimal Block Preconditioned Conjugate Gradient (LOBPCG) [4] solver present in

[§]Florida State University, School of Computational Science, cbaker@scs.fsu.edu

[¶]Sandia National Laboratories, rblehou@sandia.gov

Anasazi, operate on a matrix pencil $(A, B)$, where $A$ is symmetric and $B$ is symmetric positive-definite. Note that neither of the augmented matrices in (1.1) are positive-definite. One approach to addressing this is to ensure that the eigensolver's iteration sequence remains in a subspace where one of the matrices is positive-definite. Consider the choices from (1.1):

$$A = \begin{bmatrix} K & C^T \\ C & 0 \end{bmatrix} \quad \text{and} \quad B = \begin{bmatrix} M & 0 \\ 0 & 0 \end{bmatrix} .$$

The preconditioned eigensolvers currently operate in Salinas via the CLOP preconditioner of Clark Dohrmann. This approach works as follows. The CLOP preconditioner returns solutions of the form $\begin{bmatrix} x & y \end{bmatrix}^T$, where $Cx = 0$ and $y = 0$. The nature of these two eigensolvers is such that test subspaces generated in this manner will produce like test subspaces. As a result, the iteration sequence is always in a subspace where $B$ is positive definite so that the application of the Block Davidson and LOBPCG is well-defined. Furthermore, the $x$-component of the iterates satisfies the orthogonality constraint, so that minimizing the Rayleigh quotient for the pencil $(A, B)$ will solve the constrained eigenvalue problem (1.1).

This approach, using CLOP with the preconditioned eigensolvers, is not without its drawbacks. By explicitly enforcing the constraint and substituting zeros for the Lagrange multipliers in the iteration sequence, the method does not explicitly solve the optimality system (1.1). Therefore, the satisfaction of the constraints in the solution is entirely dependent on the level of constraint satisfaction provided by the solver. Also, because the Lagrange multipliers are fixed to zero, the direct residuals for the eigenproblem $(A, B)$ will never reach zero, because the constraints generally prevent $x$ from being an eigenvector of $(K, M)$:

$$\begin{bmatrix} K & C^T \\ C & 0 \end{bmatrix} \begin{bmatrix} x \\ 0 \end{bmatrix} - \begin{bmatrix} M & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} x \\ 0 \end{bmatrix} \lambda = \begin{bmatrix} Kx - Mx\lambda \\ 0 \end{bmatrix} .$$

This requires that the stopping conditions for these methods require modification.

**3. Optimality Characterization.** This section considers the optimality characterization for the solution of the constrained eigenvalue problem (1.2). We attempt to describe the proposed approaches to this constrained eigenvalue problem in terms of classical constrained optimization techniques (e.g., Lagrangian minimization, penalty methods). This allows us to consider a wider number of approaches for solving this problem.

Consider the formulation of the problem as the minimization of the Rayleigh quotient subject to the constraints:

$$\begin{aligned} \text{minimize} \quad & f(x) \doteq \frac{1}{2} \frac{x^T M x}{x^T K x} \\ \text{subject to} \quad & Cx = 0 . \end{aligned}$$

Note the gradient of this function:

$$\nabla f(x) \doteq \frac{1}{x^T M x} (Kx - Mx\theta_x) , \quad \theta_x = \frac{x^T K x}{x^T M x} .$$

Consider the Lagrangian of this function:

$$\mathcal{L}(x, y) = f(x) - y^T C x$$

and its gradient:

$$\nabla \mathcal{L}(x, y) = \begin{bmatrix} \nabla f(x) - C^T y \\ Cx \end{bmatrix} .$$

By considering stationary points of the Lagrangian, i.e., $\nabla\mathcal{L}(x,y) = 0$, we recover the familiar saddle-point problem:

$$\begin{bmatrix} Kx - Mx\theta_x \\ Cx \end{bmatrix} = \begin{bmatrix} C^T y \\ 0 \end{bmatrix}$$

or

$$\begin{bmatrix} K & C^T \\ C & 0 \end{bmatrix} \begin{bmatrix} x \\ -y \end{bmatrix} = \begin{bmatrix} M & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} x \\ -y \end{bmatrix} \theta_x \, .$$

A penalty method for this function could take the form

$$\phi_\rho(x) = \frac{1}{2}\frac{x^T K x}{x^T M x} + \frac{1}{2}\rho \|Cx\|^2 \, .$$

Clark Dohrmann proposed this in his note, suggesting the minimization of

$$\phi_\rho(x) = \frac{1}{2}\frac{x^T K x}{x^T M x} + \rho x^T C^T C x \, .$$

A penalty method benefits by removing the constraint on the minimization. However, as Clark noted, difficulties associated with a penalty method include the trade-off between ill-conditioning associated with large penalty coefficient $\rho$ and the need for a large $\rho$ to ensure that the minimizer satisfies the constraints.

Alternatively, consider the unconstrained minimization of the following function (from (4.2)):

$$\hat{f}(x) = \frac{1}{2}\frac{x^T (PKP^T + \rho\hat{C})x}{x^T (PMP^T + \hat{C})x} \, .$$

The analysis in Section 4 shows that the critical points of $\hat{f}$ satisfy one of the following:

1. $Cx = 0$ and $\hat{f}(x) = \lambda$, where $\lambda$ is an eigenvalue of the constrained eigenvalue problem (i.e., $(V_2^T K V_2, V_2^T M V_2)$); or
2. $Cx \neq 0$ and $\hat{f}(x) = \rho$.

If $\rho$ is greater than the largest of the targeted eigenvalues of (1.2), then an unconstrained minimization of $\hat{f}$ yields the desired solutions to the constrained eigenvalue problem (1.2). In this way, the constrained problem can be transformed into an unconstrained problem. The proposed unconstrained minimization can be solved via the eigenvalue problem (4.2).

**4. Proposed approach.** Our proposed approach reformulates the constrained eigenvalue problem (1.1) to avoid the instability of using a Krylov based eigensolver. We only assume matrix-vector products with $K$, $M$, $C$ and $N^{-1}$ (where $N$ is some suitable approximation to $K$). Our solution does not store the Lagrange multipliers. We assume that it is not feasible to compute a basis for the null space of $C$.

Multiply the eigenvalue problem (1.1) with the block diagonal matrix

$$\begin{bmatrix} S & 0 \\ 0 & I \end{bmatrix} .$$

For now, we will assume nothing about $S$ except that it is symmetric/positive definite. Later developments will explore the consequences of different choices of $S$. Application of this block diagonal preconditioner to (1.1) results in

$$\begin{bmatrix} S^{-1} & 0 \\ 0 & I \end{bmatrix} \begin{bmatrix} K & C^T \\ C & 0 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} S^{-1}K & S^{-1}C^T \\ C & 0 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} S^{-1}M & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \lambda \, .$$

Note that for a solution ($\begin{bmatrix} x & y \end{bmatrix}^T$, $\lambda$), we have

$$S^{-1}Kx + S^{-1}C^T y = S^{-1}Mx\lambda \,.$$

We wish to solve for $y$ so as to condense out the Lagrange multipliers. Pre-multiplying by $C$ and redistributing terms yields

$$CS^{-1}C^T y = CS^{-1}(M\lambda - K)x \,.$$

If $C$ has full row-rank, then $CS^{-1}C^T$ is invertible. In the case that $C$ does not have full row-rank, we must resort to its SVD-based pseudoinverse $\left(CS^{-1}C^T\right)^\dagger$ [3]. In either case, it follows that that $\left(CS^{-1}C^T\right)^\dagger C = \left(CS^{-1}C^T\right)^{-1} C$. Therefore, we can solve for $y$:

$$y = \left(CS^{-1}C^T\right)^\dagger CS^{-1}(M\lambda - K)x \,.$$

We remark then that row rank deficiency in the constraint matrix $C$ does not prevent us from continuing, though it does mean that the Lagrange multipliers $y$ corresponding to a solution $(x, \lambda)$ are not unique.

Substituting this value of $y$ into the original equation yields the equations

$$\begin{aligned} PKx &= PMx\lambda, \\ Cx &= 0 \,, \end{aligned}$$

where $P = I - \hat{C}S^{-1}$ and $\hat{C} = C^T \left(CS^{-1}C^T\right)^\dagger C$. Note that $P^T z = z$ for any $z$ satisfying $Cz = 0$, in particular our solution $x$. Then we may write

$$PKP^T x = PMP^T x\lambda \,. \tag{4.1}$$

This is a singular matrix pencil, e.g. both matrices share a null space. As a result, it defines an ill-posed eigenvalue problem. However, the singularity is not prohibitive. We now explain how to generate a symmetric positive definite matrix pencil which defines a well-posed eigenvalue problem that we can solve in a stable manner.

Note that for solutions $(x, \lambda)$ satisfying $Cx = 0$, we have $\hat{C}x = 0$. Then we note the following progression:

$$(PKP^T + \rho\hat{C})x = PKP^T x = PMP^T x\lambda = (PMP^T + \hat{C})x\lambda \,. \tag{4.2}$$

For a positive $\rho$, shifting in this manner produces a symmetric/positive definite eigenvalue problem. The finite eigenvalues from (4.1) are eigenvalues of (4.2), and their associated eigenvectors satisfy the problem constraint. The remaining eigenvalues of (4.2) take the value $\rho$, which can be chosen larger than the eigenvalues of interest.

We now discuss approaches for solving the new eigenvalue problem:

$$(PKP^T + \rho\hat{C})x = (PMP^T + \hat{C})x\lambda \,.$$

A Krylov subspace solver requires a spectral transformation, which requires that we solve (for $w$) linear systems of the form

$$(PKP^T + \rho\hat{C})w = (PMP^T + \hat{C})z \,.$$

The solution to this system is not immediately apparent.

Other alternatives include preconditioned residual-based solvers, such as Block Davidson or LOBPCG (both available in Anasazi). These solvers as a fundamental step precondition the eigenvector residual. Consider some preconditioner $N \approx K$. Then the preconditioned residual for a Ritz pair $(x, \lambda)$ is

$$N^{-1}r = N^{-1}P(K - M\lambda)P^T x + N^{-1}(\rho - 1)\hat{C}x \, .$$

The result of this computation depends on the choice of $S$ from above. One choice immediately presents itself: $S = N$. In this case, we have two useful identities: $P = I - \hat{C}N^{-1}$ and $N^{-1}P = P^T N^{-1}$. This means that $N^{-1}r$ can be rewritten as

$$N^{-1}r = P^T N^{-1}(K - M\lambda)P^T x + (\rho - 1)(I - P^T)x \, .$$

Note in particular that for $x$ already satisfying $P^T x = x$ (i.e., $Cx = 0$), the second term in this equation becomes zero and $N^{-1}r = P^T N^{-1}(K - M\lambda)x$, which also satisfies $P^T N^{-1}r = N^{-1}r$. This is a useful property for the Block Davidson solver. This is because the successive subspaces (from which a solution is extracted) are expanded by $N^{-1}r$. As a result, if the search subspace is already orthogonal to $C^T$, then it will remain so after the expansion by $N^{-1}r$. A similar recurrence applies to LOBPCG as well.

## REFERENCES

[1] P. Arbenz, U. L. Hetmaniuk, R. B. Lehoucq, and R. S. Tuminaro, *A comparison of eigensolvers for large-scale 3D modal analysis using AMG-preconditioned iterative methods*, International Journal for Numerical Methods in Engineering, 64 (2005), pp. 204–236.

[2] M. Bhardwaj, K. Pierson, G. Reese, T. Walsh, D. Day, K. Alvin, and J. Peery, *Salinas: A scalable software for high-performance structural and solid mechanics simulation*.

[3] G. H. Golub and C. F. Van Loan, *Matrix Computations, third edition*, Johns Hopkins Studies in the Mathematical Sciences, Johns Hopkins University Press, 1996.

[4] A. Knyazev, *Toward the optimal preconditioned eigensolver: Locally optimal block preconditioned conjugate gradient method*, SIAM J. Sci. Comput., 23 (2001), pp. 517–541.

[5] R. Lehoucq, D. Sorensen, and C. Yang, *Arpack users' guide: Solution of large scale eigenvalue problems with implicitly restarted Arnoldi methods*, 1997.

[6] K. J. Maschhoff and D. C. Sorensen, *PARPACK: An efficient portable large scale eigenvalue package for distributed memory parallel architectures*, Lecture Notes in Computer Science, 1184 (1996), pp. 478–486.

## Discrete Mathematics and Informatics

Discrete mathematics is the study of fundamentally discrete mathematical structures. This particular branch of mathematics is strongly applicable to applications arising in the computing sciences, due to the discrete nature of computation. Correspondingly, the field of informatics includes processing and reasoning about collected information or data, and can be considered as encompassing the whole of computer science and related fields. The articles in this section make contributions in these broad areas.

Wolf and Boman explore new partitioning techniques to improve scalability of parallel, sparse matrix-vector multiplication, a core computational kernel for large-scale simulations. Selee *et al.* consider the problem of how to group information when multiple similarities are known. To this end they develop a new tensor decomposition they call the Implicit Slice Canonical Decomposition (IMSCAND) and demonstrate the applicability of IMSCAND on a set of journal articles with multiple similarities. Finally, Benavides *et al.* introduce a new Python package, Pyomo (Python Optimization Modeling Objects). Pyomo provides capabilities similar to those of other algebraic modeling languages (AMLs), which are high-level programming languages for describing and solving mathematical problems, particularly optimization problems. Pyomo can be used to define abstract problems, create concrete problem instances, and solve these instances with standard solvers.

M.L. Parks
S.S. Collis

December 6, 2007

# PARTITIONING FOR PARALLEL SPARSE MATRIX-VECTOR MULTIPLICATION

MICHAEL M. WOLF[*] AND ERIK G. BOMAN[†]

**Abstract.** Parallel sparse matrix-vector multiplication is ubiquitous throughout large-scale scientific simulations. As simulations grow to tens of thousands of processors and higher, the communication volume will become increasingly significant. In order to mitigate this growing communication volume, we must utilize more complicated partitioning techniques than traditionally necessary. In this paper, we will outline previous partitioning methods and introduce a new method we have developed.

## 1. Introduction.

**1.1. Motivation.** Parallel computing is essential to modern computational science. An important motivating factor for parallel computing is large-scale scientific simulations. These simulations are often too large to fit in memory on one computer and take too long to compute in serial. Thus, the computation and often the data must be distributed across multiple processors so that the computational scientist can have their simulation complete in a timely manner. For matrix-vector multiplication, this means distributing both the matrices and the vectors across the processors. Figure 1.1 shows a possible distribution of both vectors and matrix nonzeros for the matrix-vector multiplication operation $\mathbf{y} = \mathbf{Ax}$ with the different colors representing different partitions. For this paper, we will assume that the input and output vectors are distributed identically (generally a good assumption) and that the partition of each vector entry ($x_i$ and $y_i$) is the same as the partition of the corresponding diagonal entry in the matrix, $a_{i,i}$. Figure 1.2 shows a different representation of the same parallel matrix-vector product, which is useful in visualizing the communication volume for this operation. Since for partitioning the actual value of a nonzero is not important (only the fact that the element is a nonzero is important), we have replaced the nonzero values of the matrix with X's. Again, the color of the X's corresponds to a particular partition. We have also replaced the vectors with segmented bars where the entries are colored by partition. We align the **y** color bar to the left of the matrix so that each entry in the **y** color bar is directly to the left of the matrix row whose inner-product calculates this entry. We align each **x** color bar entry directly above the matrix column entries with which they are multiplied in the matrix-vector product. This alignment makes it easier to visualize the communication needed for the matrix-vector product as described in subsection 1.2.

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \\ y_8 \end{bmatrix} = \begin{bmatrix} 1 & 6 & 0 & 0 & 0 & 0 & 0 & 0 \\ 5 & 1 & 9 & 0 & 5 & 0 & 0 & 0 \\ 8 & 0 & 1 & 7 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 1 & 0 & 0 & 0 & 7 \\ 0 & 0 & 0 & 0 & 1 & 8 & 0 & 0 \\ 0 & 4 & 0 & 0 & 3 & 1 & 3 & 0 \\ 0 & 0 & 0 & 6 & 0 & 9 & 1 & 4 \\ 0 & 0 & 0 & 0 & 0 & 0 & 2 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \end{bmatrix}$$

FIG. 1.1. *Distribution of matrices and vectors for parallel sparse matrix-vector multiplication.*

[*]University of Illinois at Urbana-Champaign, mmwolf@uiuc.edu
[†]Sandia National Laboratories, egboman@sandia.gov

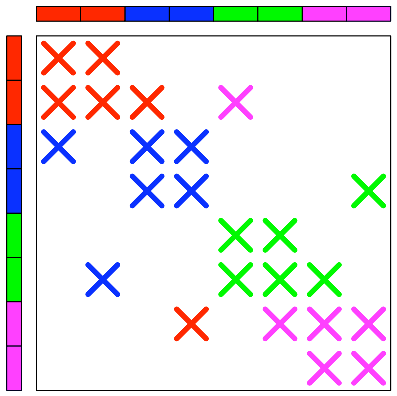


FIG. 1.2. *Alternative way of visualizing parallel sparse matrix-vector multiplication.*

**1.2. Parallel Matrix-Vector Multiplication.** In general, there are four main stages of parallel matrix-vector multiplication as shown in Figure 1.3 and summarized in the following enumeration:

1. **Expand:** Send entries $x_j$ to processes with a nonzero $a_{i,j}$ for some row $i$.
2. **Local Multiply-add:** $y_i := y_i + a_{i,j}x_j$
3. **Fold:** Send partial inner-product (**y** values) to relevant processes.
4. **Sum:** Sum up the partial **y** values.

In the first stage, elements in vector **x** are communicated to remote processes. In particular, $x_j$ is communicated to a remote process if that process owns a nonzero in the $j$th column of matrix **A** as shown in Figure 1.3(a). From this diagram, we can easily determine that communication is needed if there is a nonzero in a column of a different color than the color of the **x** element for that column. For example, since $x_1$ is owned by the red process but $a_{3,1}$ is a blue process nonzero, $x_1$ must be communicated to the blue process. After this first communication stage, the processes perform local partial inner-product operations for the nonzeros that they own (Figure 1.3(b)). Next, each process communicates the partial inner-product results to the processes which own the corresponding **y** entry. From the diagram in Figure 1.3(c), we can easily determine that communication is needed if a nonzero in row $i$ is a different color than $y_i$. For example, since $y_4$ is owned by the blue process but $a_{4,8}$ is owned by the green process, the local partial inner-product $\hat{y}_4 := a_{4,8}x_8$ must be communicated from the green process to the blue process. Finally, the processes accumulate the partial inner-products to form the vector entries of **y** (Figure 1.3(d)).

When partitioning for parallel matrix-vector multiplication, we are interested in reducing the actual run-time of the algorithm. We could write an objective function to minimize the run-time, taking into consideration computation, communication latency, communication volume, idle time, etc. However, this would would be a very difficult optimization problem with so many contributing variables to solve in a reasonable amount of time. Thus, in practice, we settle for minimizing the total communication volume while keeping the computation balanced across processes. When partitioning to minimize this objective, we can use either one-dimensional partitioning (section 2) or two-dimensional partitioning (section 3). We can also model the communication in several different ways, using graphs, bipartite graphs, or hypergraphs, for example. In the following sections, we discuss one-dimensional and two-dimensional methods using graphs and hypergraphs.

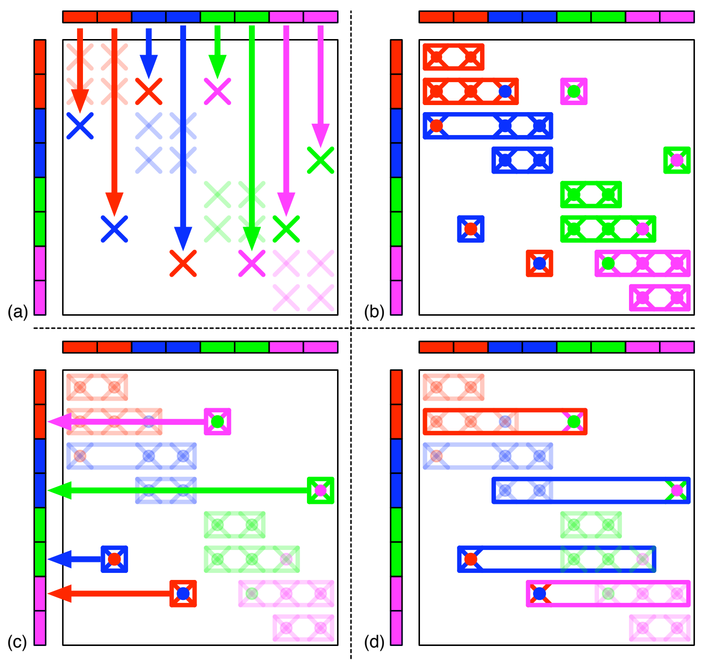


FIG. 1.3. *Stages of parallel sparse matrix-vector multiplication.*

**2. One-Dimensional Partitioning.** One-dimensional partitioning can either be one-dimensional row partitioning or one-dimensional column partitioning. In one-dimensional row partitioning, each process is assigned all the nonzeros for some set of rows (Figure 2.1(a)). Similarly, in one-dimensional column partitioning, each process is assigned all the nonzeros for some set of columns (Figure 2.1(b)). A parallel matrix-vector multiplication operation resulting from one-dimensional partitioning only has one communication stage. In particular, for an operation resulting from one-dimensional row partitioning, the partial inner-products need not be communicated since a process that owns a particular row also owns the corresponding **y** vector entry. Likewise, for an operation resulting from one-dimensional column partitioning, the **x** vector entries do not need to be communicated since a process that owns a particular nonzero also owns the corresponding **x** entry by which it is multiplied during the local inner-product stage.

**2.1. One-Dimensional Graph Model.** One frequently utilized model of communication is the one-dimensional graph model (as shown in Figure 2.2) [9, 10]. For this model, we assume the matrix is symmetric. Each matrix row or column (depending on whether row or column partitioning is requested) is represented by a vertex in the graph. The off-diagonal nonzeros are represented by edges between the two vertices corresponding to the row and column of the nonzero. For instance, in Figure 2.2, element $a_{1,8}$ is a nonzero, and thus vertices 1 and 8 are connected by an edge. After constructing the graph, we partition the vertices into $k$ equal sets ($k = 2$ for Figure 2.2) such that the number of cut edges is minimized. A cut edge is an edge that connects two vertices of different partitions. The graph model estimates the communication volume to be twice the number of cut edges. This partitioning of the graph model is NP-hard to solve optimally. However, there are many heuristic algorithms that can solve this problem close to optimally in polynomial time [9, 10].

However, there are a couple of drawbacks to using this traditional graph model. The

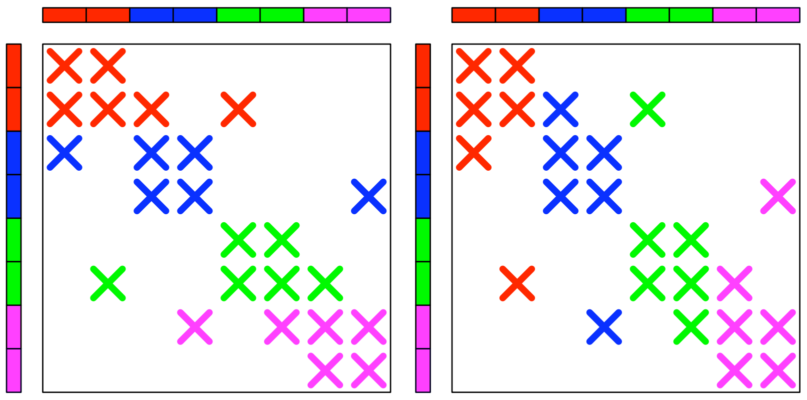

(a) Row partitioning. (b) Column partitioning.

FIG. 2.1. *One-dimensional partitioning.*

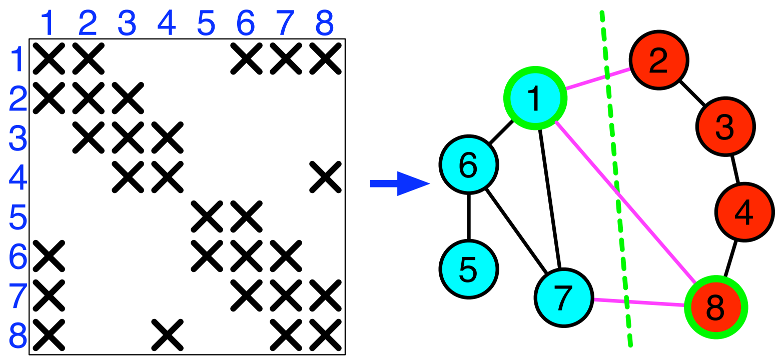


FIG. 2.2. *One-dimensional graph partitioning.*

graph model requires the matrix to have a symmetric nonzero structure. A more severe problem is that using twice the edge cut as an metric for the communication volume is not accurate. In particular, we see in the Figure 2.2 graph that the communication volume is over-counted. For this partitioning, there are three cut edges (highlighted in magenta): {1,2}, {1,8}, and {7,8}. Using the metric, we get a communication volume of 6. However, the cut edges involving vertices 1 and 8 are over-counted by this metric since the vertices only should be communicated once, and the true communication volume for this matrix-vector product should be 4. The over-counting in the graph model can be remedied by counting boundary vertices instead. However, most people that use graph partitioning do not use this more correct boundary vertex version but use the traditional edge cut version. For some applications, e.g. structured meshes, the difference between the edge cut and bounding vertices is small, and thus the error is also small.

**2.2. One-Dimensional Hypergraph Model.** A model that addresses the shortcomings of the one-dimensional graph model (using the edge cut metric) is the one-dimensional hypergraph model (shown in Figure 2.3 for row partitioning). Unlike the graph model, the hypergraph model allows for matrices with unsymmetric nonzero patterns. For the one-dimensional row hypergraph partitioning, the rows are represented by vertices in the hypergraph (for one-dimensional column partitioning, columns are represented by vertices). Each column is represented by a hyperedge in the hypergraph. For instance, in Figure 2.3, the third column of the matrix has nonzeros in rows 2, 3, and 4. Thus, the corresponding hyperedge in the hypergraph contains vertices 2, 3, and 4. A typical representation of the hypergraph model is shown in the right diagram of Figure 2.3. However, we can also visualize the hypergraph directly on the matrix stencil (left diagram of Figure 2.3), with the hyperedges drawn on the matrix rows or columns (columns for the row partitioning shown in the figure). For the one-dimensional row partitioning, each nonzero in a row corresponds to the same vertex in the hypergraph, and thus we can obtain the right diagram by superimposing the matrix columns for the left diagram and rearranging the X's into the same positions as the vertices on the right. After constructing the hypergraph, we partition the vertices into $k$ equal sets ($k = 2$ for Figure 2.3) such that a hyperedge cut metric is minimized. The cut metric is obtained by summing over all hyperedges the number of different remote processes (those that do not own the diagonal entry) owning vertices for a given hyperedge. Aykanat and Catalyurek proved that this hyperedge cut metric is equivalent to the total communication volume [2]. For the hypergraph shown in Figure 2.3, there are two cut hyperedges (the column 3 blue and column 5 cyan shaded hyperedges) and thus a communication volume of two for the resulting matrix-vector product, which is accurate for this partitioning of the matrix. As with the graph model, partitioning of the hypergraph is NP-hard to solve optimally but there are heuristic algorithms that can solve this problem close to optimally in polynomial time [2, 7].

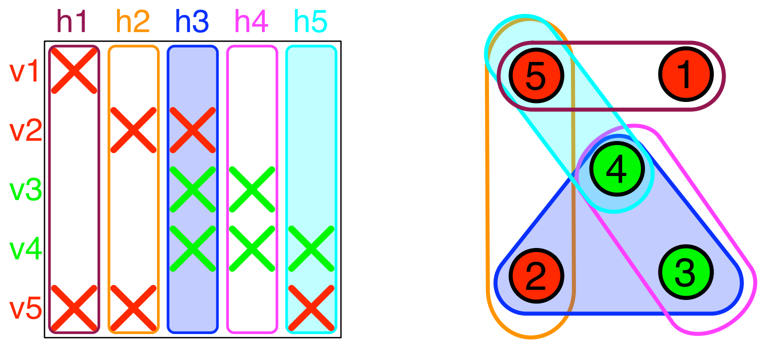


FIG. 2.3. *One-dimensional hypergraph partitioning.*

**2.3. Inadequacy of One-Dimensional Partitioning.** One-dimensional sparse matrix partitioning is sufficient for many problems, and most applications use matrices distributed in a one-dimensional manner. However, for some problems one-dimensional partitioning is potentially disastrous in terms of the communication volume. The "arrowhead" matrix shown in Figure 2.4 is an example for which one-dimensional partitioning is inadequate. For the bisection ($k = 2$) case shown in the figure, any load-balanced one-dimensional partitioning will yield a communication volume of approximately $\frac{3}{4}n$ for the matrix-vector product. As we will see in the following sections, this is far from a minimum communication volume for this problem and it is unacceptable for the communication volume to scale as $n$ for this matrix. Thus, we need more flexible partitioning than traditional one-dimensional partitioning.

**3. Two-Dimensional Partitioning.** Two-dimensional partitioning is a more flexible alternative to one-dimensional partitioning, in which there is no specific partition assigned to a given row or column. Thus, we have to specify the partition for particular sets of nonzeros. Two-dimensional Cartesian partitioning is a simple method of two-dimensional partitioning in which a partition is assigned to the nonzeros which lie in both a particular set of rows and

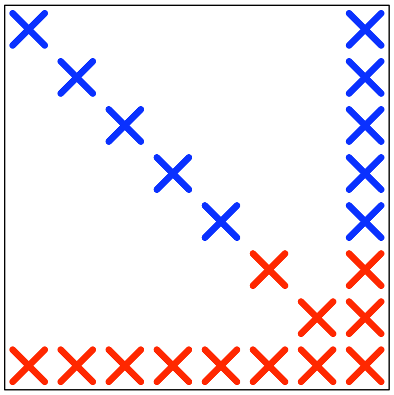


FIG. 2.4. *Arrowhead matrix partitioned for two processes.*

a particular set of columns. This partitioning is obtained by partitioning the matrix into $\sqrt{k}$ processes in one dimension, say row-wise, and then partitioning each of the row partitions into $\sqrt{k}$ column partitions for a total of $k$ partitions. Figure 3.1 shows a block version of this method where the partitions consist of nonzeros in a set of continuous rows and columns. Although Cartesian block partitioning is a good method for dense matrices, it suffers from potential poor load-balancing for most sparse matrices. Although slight improvements can be made by using one-dimensional hypergraph partitioning in both directions to obtain a more scattered Cartesian partitioning [4], the method still in general suffers from poor load-balancing.

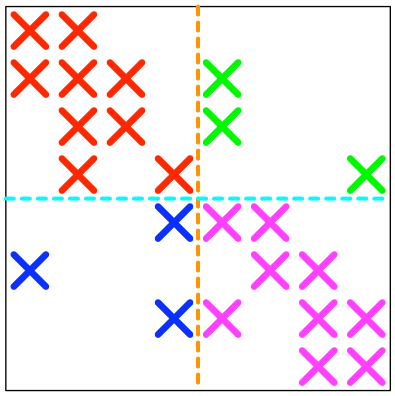


FIG. 3.1. *Two-dimensional Cartesian partitioning.*

**3.1. Mondriaan.** A slightly more general and flexible two-dimensional partitioning method is the Mondriaan method [11]. Mondriaan uses recursive bisection such that at each level of the algorithm the partitions from the previous level can be partitioned by either rows or columns. As shown in Figure 3.2, this yields a rectangular tiled partitioning where each partition tile can have varied dimensions. In Figure 3.2, we see that the first level partitioning was made row-wise (division shown by the cyan line). The second level partitioning was made column-wise for the top portion but row-wise for the lower partition (orange lines). As with the Cartesian method, the partitions need not consist of consecutive rows/columns but were shown this way for easier illustration.

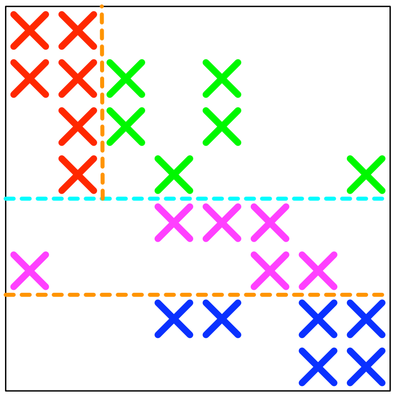


Fig. 3.2. *Two-dimensional Mondriaan partitioning.*

**3.2. Fine-grain hypergraph.** The most flexible partitioning method is the fine-grain hypergraph partitioning method in which each nonzero can be partitioned separately from the others [3]. In the fine-grain hypergraph model, each nonzero is assigned a partition separately and thus is represented by a vertex in the hypergraph. Each row is represented by a hyperedge in the hypergraph (magenta hyperedges in Figure 3.3). Likewise, each column is represented by a hyperedge in the hypergraph (orange hyperedges in Figure 3.3). Thus, for a $n \times n$ matrix, the fine-graph hypergraph model has $2n$ hyperedges.

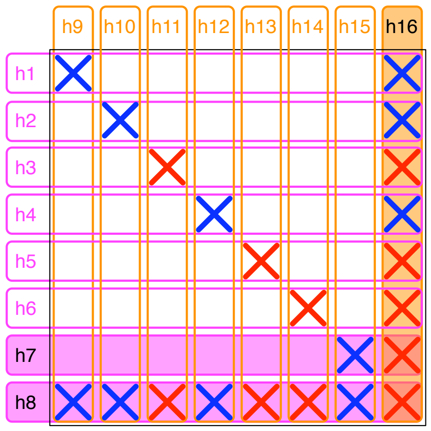


Fig. 3.3. *Fine-grain hypegraph partitioning of arrowhead matrix with $k = 2$ partitions. Cut hyperedges are shaded. The hyperedge cut and thus the communication volume are 3.*

As with the one-dimensional hypergraph model, we partition the vertices into $k$ equal sets ($k = 2$ in Figure 3.3) such that the hypergraph cut metric described in subsection 2.2 is minimized. Again, the communication volume is equivalent to this hyperedge cut metric. Catalyurek and Aykanat proved that this fine-grain hypergraph model yields a minimum volume partitioning when optimally solved [3]. In Figure 3.3, we see the fine-graph hypergraph partitioning of the $8 \times 8$ arrowhead matrix. The resulting communication volume is shown to be 3, which is a significant improvement over the communication volume of 6 from the optimal one-dimensional partitioning. As with the one-dimensional hypergraph model, solving the fine-grain hypergraph model optimally is NP-hard but there are heuristics that can solve it close to optimally in polynomial time. Unfortunately, the resulting fine-grain hypergraph problem is a larger NP-hard problem and thus, may be too expensive to solve quickly for large matrices.

## 4. Two-Dimensional Corner Partitioning.

**4.1. Motivation.** Loosening the load-balancing restriction slightly so that the number of nonzeros are allowed to differ slightly between partitions, we obtain the fine-grain hypergraph partitioning ($k = 2$) shown in Figure 4.1 for the $8 \times 8$ arrowhead matrix. This partitioning will result in a communication volume of 2, which is the minimum cut/volume possible for any non-trivial partitioning. An examination of this minimum cut partitioning suggests a new partitioning method. We see that each partition consists of a set of ""corners" (more easily seen in Figure 4.2), which are basically one-dimensional partitions reflected across the diagonal. Using these ""corners", the hope is that we could reproduce an optimal fine-grain partitioning using a less costly one-dimensional partitioning method for certain matrices.

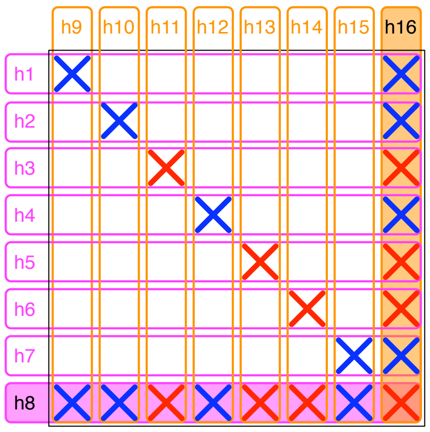


Fig. 4.1. *Fine-grain hypergraph partitioning ($k = 2$) with slight imbalance yielding minimum hyperedge cut for non-trivial partitioning.*

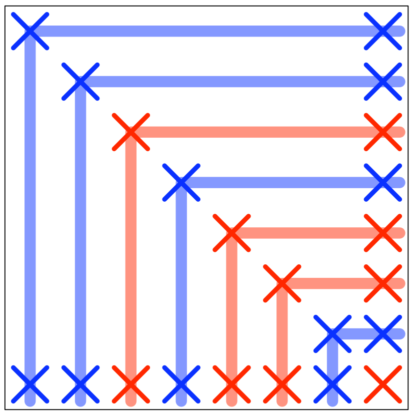


Fig. 4.2. *"Corners" in Figure 4.1 partitioning.*

**4.2. Method.** We show an illustration of the corner partitioning method in Figure 4.3 for the previously partitioned $8 \times 8$ arrowhead matrix. As shown in the diagrams of Figure 4.3(a,b), we start with a one-dimensional column partitioning of the lower triangular part of the matrix. We then reflect this one-dimensional partitioning across the diagonal such that row $i$ in the upper triangular part of the matrix is assigned to the same partition as column $i$ in the lower triangular part of the matrix (Figure 4.3(c)). For this arrowhead matrix, we see in Figure 4.3(d) that this corner partitioning method has produced the same optimal partitioning as obtained by the fine-grain hypergraph method (Figure 4.1) at a reduced computational cost. This indicates that the corner method can be an effective two-dimensional partitioning method for some matrices.

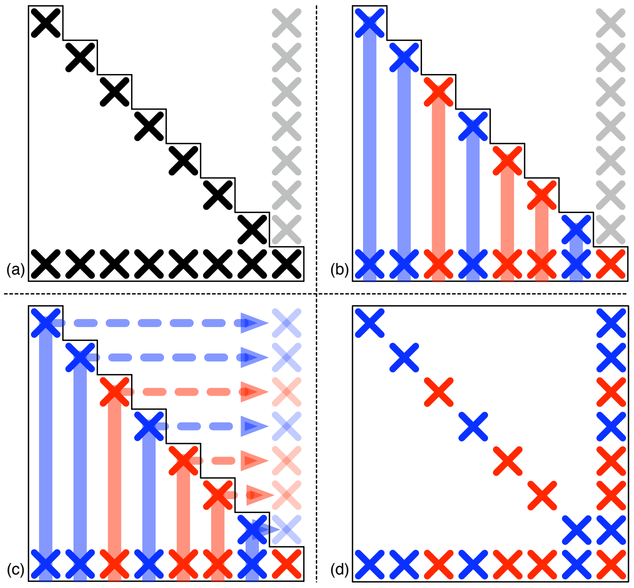


Fig. 4.3. *Corner partitioning method.*

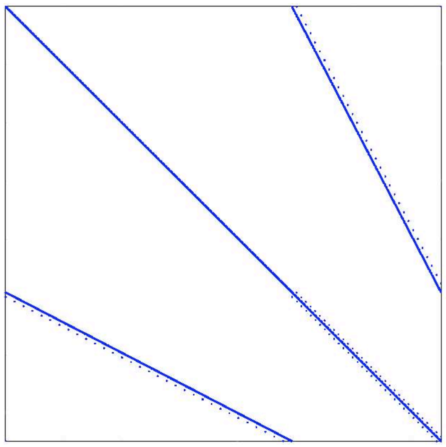
**5. Results.** We implemented one-dimensional column, two-dimensional corner, and two-dimensional fine-grain partitioning by using the hypergraph partitioning algorithm from the Zoltan library [1, 6, 7]. We studied the partitioning of three different matrices: an arrowhead matrix and two "real world" matrices from the literature. We also obtain results for two-dimensional Mondriaan partitioning method from the literature [11] and by running the Mondriaan code [12]. We compare the resulting communication volumes obtained by these four methods for the three test matrices.

The arrowhead matrix we studied had $n = 40000$ rows and columns. We summarize the resulting communication volumes for the methods in Table 5.1. As expected, the one-dimensional column method does a very poor job of partitioning the arrowhead matrix. The resulting communication volume is approximately $\frac{3}{4}n$ for the bisection case and approximately $n$ for the larger number of partitions. The two-dimensional Mondriaan method also partitioned the arrowhead matrix poorly. This is not too surprising when we consider that the first cut of this multi-level algorithm is one-dimensional bisection. After this first cut, the communication will only increase with additional cuts. Thus, although Mondriaan is slightly better than one-dimensional column partitioning for the larger numbers of partitions, the first cut has doomed this method to yield a high communication volume partitioning. The corner method and fine-grain hypergraph method both yield significant better results than the other two methods. They yield partitionings with minimum communication volume (for non-trivial partitioning), $2(p-1)$, which is much improved over the order $n$ volumes of the other methods since usually $p \ll n$. Thus, the corner method performed well for this matrix, yielding the same quality partitioning as the fine-grain hypergraph model at a cheaper cost.

Table 5.1
*Partitioning results for* $40000 \times 40000$ *arrowhead matrix for four methods: one-dimensional column, two-dimensional Mondriaan, two-dimensional corner, and fine-grain hypergraph partitioning. Each entry gives the communication volume resulting from for a method's partitioning for* $k = 2, 4, 16, 64$ *partitions. The starred entries designate a minimum volume for a non-trivial partitioning.*

| k | 1-D Column | Mondriaan | Corner | Fine-grain |
|---|---|---|---|---|
| 2 | 29101 | 29102 | **2***| **2*** |
| 4 | 40001 | 29778 | **6*** | **6*** |
| 16 | 40012 | 37459 | **30*** | **30*** |
| 64 | 40048 | 39424 | **126*** | **126*** |

Since we possessed Mondriaan results from the literature [11] for **finan512** and **bcsstk30**, we used these matrices to compare the partitioning methods. The **finan512** matrix, which we obtained from the University of Florida sparse matrix collection [5], resulted from portfolio optimization. The nonzero structure is shown in Figure 5.1(a). The **bcsstk30** matrix is from the Harwell-Boeing collection and arose from a structural engineering eigenvalue problem [8]. The nonzero structure for **bcsstk30** is shown in Figure 5.1(b).


(a) finan512 matrix.

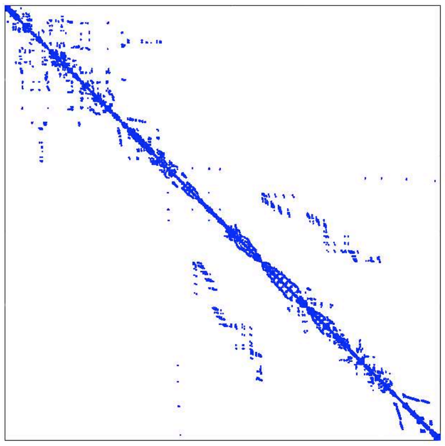

(b) bcsstk30 matrix.

Fig. 5.1. *Nonzero patterns of test matrices.*

As with the arrowhead matrix, we partitioned these two matrices using the one-dimensional, two-dimensional corner, and two-dimensional fine-grain hypergraph methods for 2, 4, 16, and 64 partitions. We obtained results for the two-dimensional Mondriaan method from the literature [11]. Figure 5.2 shows the results for the **finan512** matrix. Similar to the arrowhead matrix, the fine-grain and corner methods yield significantly more optimal partitionings than the one-dimensional and Mondriaan methods for the higher number of partitions. The corner method actually produce slightly lower communication volume partitionings than the fine-grain hypergraph method at a reduced cost. Figure 5.3 plots the communication volume resulting from the partitioning methods for the **bcsstk30** matrix. Again, we see that the corner method yields the partitioning of the lowest communication volume for this problem, especially for the higher number of partitions. Interestingly, the fine-grain hypergraph method yields the highest communication volume partitioning. This may be due to the heuristic partitioning algorithms having difficulty optimizing this larger hypergraph problem.

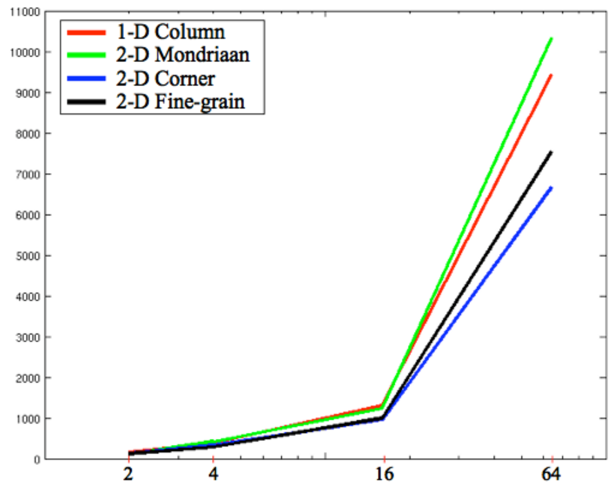


FIG. 5.2. *finan512 matrix: communication volume for four partitioning methods for* $k = 2, 4, 16, 64$ *partitions.*

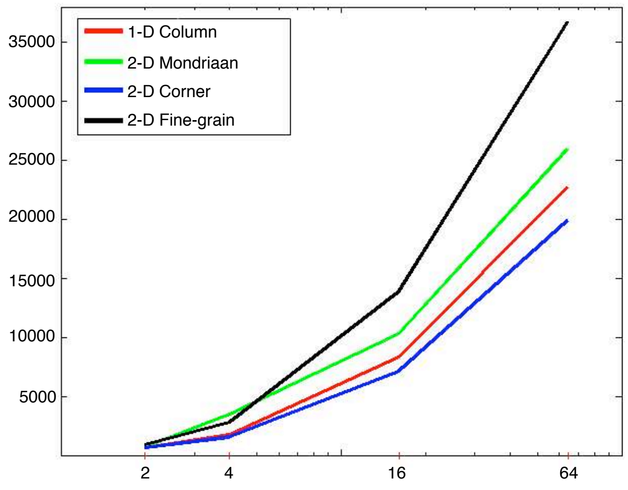


FIG. 5.3. *bcsstk30: communication volume for four partitioning methods for* $k = 2, 4, 16, 64$ *partitions.*

**6. Summary/Conclusions.** We have outlined several methods of partitioning matrices for matrix-vector multiplication, including both one-dimensional and two-dimensional partitioning methods. In subsection 2.3, we described a particular matrix for which one-dimensional partitioning yielded poor partitioning results and argued that a more flexible two-dimensional partitioning scheme was necessary. We introduced a new method of two-dimensional matrix partitioning, the corner method. As hoped, we showed that this corner method could produce partitionings of similar quality (better for some matrices) to the fine-grain hypergraph method at a reduced cost.

In the future, we wish to gain a better intuition for the corner partitioning method. It is clear that it produces an optimal (non-trivial) partitioning for the arrowhead matrix and that it produces very good partitionings for both the **finan512** and the **bcsstk30** matrices. However, we would like to have a better intuition of for what matrices the corner method produces good partitionings. In more recent work, we have been studying symmetric reordering of the matrix rows and columns for the corner partitioning method. Reordering is unnecessary for one-dimensional partitioning schemes since it yields the same graph and hypergraph models (although in practice these may differ when the problem is solved less than optimally using heuristics). However, the corner symmetric partitioning method is very dependent on the row/column ordering. Thus, reordering can potentially greatly decrease the communication volume for a corner partitioning. We would like to be able to find the optimal ordering/partitioning for the corner method and hope that this will extend the utility of the method so that it will be useful for partitioning a wider variety of matrices.

**Acknowledgments.** We thank Bruce Hendrickson for his recent contributions on the symmetric row/column reordering for the corner method. We would also thank the Zoltan development team since we used Zoltan for most of our partitioning. This research was performed as part of the CSCAPES Institute, funded by the Office of Science of the U.S. Department of Energy under the Scientific Discovery through Advanced Computing (SciDAC) program.

## REFERENCES

[1] E. Boman, K. Devine, L. A. Fisk, R. Heaphy, B. Hendrickson, V. Leung, C. Vaughan, U. Catalyurek, D. Bozdag, and W. Mitchell, *Zoltan home page*. `http://www.cs.sandia.gov/Zoltan`, 1999.

[2] Ü. Çatalyürek and C. Aykanat, *Hypergraph-partitioning-based decomposition for parallel sparse-matrix vector multiplication*, IEEE Trans. Parallel Dist. Systems, 10 (1999), pp. 673–693.

[3] ———, *A fine-grain hypergraph model for 2d decomposition of sparse matrices*, in Proc. IPDPS 8th Int'l Workshop on Solving Irregularly Structured Problems in Parallel (Irregular 2001), April 2001.

[4] ———, *A hypergraph-partitioning approach for coarse-grain decomposition*, in Proc. Supercomputing 2001, ACM, 2001.

[5] T. A. Davis. The University of Florida Sparse Matrix Collection, 1994. `http://www.cise.ufl.edu/research/sparse/matrices/`.

[6] K. Devine, E. Boman, R. Heaphy, B. Hendrickson, and C. Vaughan, *Zoltan data management services for parallel dynamic applications*, Computing in Science and Engineering, 4 (2002), pp. 90–97.

[7] K. D. Devine, E. G. Boman, R. T. Heaphy, R. H. Bisseling, and U. V. Catalyurek, *Parallel hypergraph partitioning for scientific computing*, IEEE, 2006.

[8] I. S. Duff, R. G. Grimes, and J. G. Lewis, *Sparse matrix test problems*, ACM Trans. Mathematical Software, 15 (1989), pp. 1–14.

[9] G. Karypis and V. Kumar, *A fast and high quality multilevel scheme for partitioning irregular graphs*, SIAM J. Scientific Computing, 20 (1998), pp. 359–392.

[10] R. Leland and B. Hendrickson, *An empirical study of static load balancing algorithms*, in Proc. Scalable High Perf. Comput. Conf., IEEE, May 1994, pp. 682–685.

[11] B. Vastenhouw and R. Bisseling, *A two-dimensional data driven distribution method for parallel sparse matrix-vector multiplication*, SIAM Review, 47 (2005), pp. 67–95.

[12] B. Vastenhouw and R. H. Bisseling, *Mondriaan software version 1.0.2*, 2005. `http://www.math.uu.nl/people/bisseling/Mondriaan/mondriaan.html`.

# EXTRACTING CLUSTERS FROM LARGE DATASETS WITH MULTIPLE SIMILARITY MEASURES USING IMSCAND

TERESA M. SELEE[*], TAMARA G. KOLDA[†], W. PHILIP KEGELMEYER[‡], AND JOSHUA D. GRIFFIN[§]

**Abstract.** We consider the problem of how to group information when multiple similarities are known. For a group of people, we may know their education, geographic location and family connections and want to cluster the people by treating all three of these similarities simultaneously. Our approach is to store each similarity as a slice in a tensor. The similarity measures are generated by comparing features. Generally, the object similarity matrix is dense. However it can be stored implicitly as the product of a sparse matrix, representing the object-feature matrix, and its transpose. For this new type of tensor where dense slices are stored implicitly, we have created a new decomposition called Implicit Slice Canonical Decomposition (IMSCAND). Our decomposition is equivalent to the tensor CANDECOMP/PARAFAC decomposition, which is a higher-order analogue of the matrix Singular Value decomposition (SVD) and Principal Component Analysis (PCA). From IMSCAND we obtain compilation feature vectors which are clustered using *k*-means. We demonstrate the applicability of IMSCAND on a set of journal articles with multiple similarities.

## 1. Introduction.

Datasets naturally have mulitple similarities. For a group of people, we might know their age, education, geographic location, and social and family connections. For a set of published papers, we know the authors, citations, and terms in the abstract, title, and keywords. Even for a computer hard drive, we know the names of the files, their saved location, their time stamp, and their contents. For each of these examples, if we wanted to find a way to cluster the people, documents, or computer files, our approach is to treat all the similarities concurrently.

For each similarity, a similarity matrix is formed, with objects (people, documents, files, etc.) as both the rows and columns. Each of the similarity matrices is a slice in a tensor, and a tensor decomposition is used to assemble the multiple adjacency matrices into a set of compilation feature vectors. These feature vectors can then be clustered using the *k*-means clustering algorithm. This approach was used by Dunlavy et al. [6].

Our approach to this problem is special because of how we form the similarity matrices. In general, similarity matrices are dense, limiting the number of objects and features because of the large amount of work required to compute a decomposition for a dense tensor. We form the dense similarity matrices implicitly by storing only sparse object-feature matrices. For example, an article-author matrix is a sparse object-feature matrix with documents as rows and authors as columns of the matrix. Since most articles have just a few authors, we can see how this is a sparse matrix. Then the adjacency matrix is stored implicitly as the product of the sparse object-feature matrix and its transpose (a feature-object matrix).

Our new decomposition is called Implicit Slice Canonical Decomposition (IMSCAND) and does all of its computations on the sparse matrices only. This allows us to treat much larger problems than if we stored the full similarity matrices. We illustrate the effectiveness of our decomposition on a set of journal publications from the Society of Industrial and Applied Mathematics (SIAM). Our long-term goal is to use this approach to cluster files on computer hard drives.

## 2. Tensors.

In this paper we focus on third-order tensors. These are denoted by boldface Euler script letters, e.g., $\mathcal{X}$. When we say order, way, or mode we are referring to the number of dimensions of the tensor. A third-order tensor, $\mathcal{X} \in \mathbb{R}^{I \times J \times K}$ is illustrated in Figure 2.1.

[*]Department of Mathematics, North Carolina State University, tmselee@ncsu.edu
[†]Sandia National Laboratories, tgkolda@sandia.gov
[‡]Sandia National Laboratories, wpk@sandia.gov
[§]Sandia National Laboratories, jdgriffi@sandia.gov

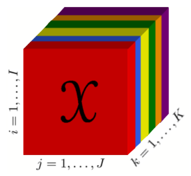


FIG. 2.1. *Third-order tensor* $\mathcal{X} \in \mathbb{R}^{I \times J \times K}$

Matrices are denoted by boldface capital letters, e.g., $\mathbf{A}$. Vectors are denoted by boldface lowercase letters, e.g., $\mathbf{a}$. Scalars are denoted by lowercase letters, e.g., $a$. The $i^{th}$ entry of $\mathbf{a}$ is $a_i$. The $i^{th}$ row of $\mathbf{A}$ is $\mathbf{a}_{i:}$, the $j^{th}$ column of $\mathbf{A}$ is $\mathbf{a}_{:j}$ or sometimes just $\mathbf{a}_j$. Finally, the $(i, j)$ entry of $\mathbf{A}$ is $a_{ij}$.

We have similar notation for tensors. The higher-order analogue of rows and columns are fibers. A column vector is a mode-1 or column fiber, denoted $\mathbf{x}_{:jk}$. A row vector is a mode-2 or row fiber, denoted $\mathbf{x}_{i:k}$. Vectors in the third dimension are tube fibers, written $\mathbf{x}_{ij:}$. These are illustrated in Figure 2.2.

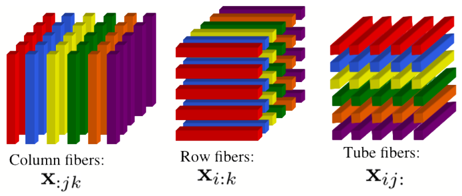


FIG. 2.2. *Fibers of a third order tensor* $\mathcal{X}$.

We must also discuss the notation for two-dimensional slices. Slices of a tensor are matrices and can be horizontal, lateral, or frontal. They are denoted as $\mathbf{X}_{i::}, \mathbf{X}_{:j:}$ and $\mathbf{X}_{::k}$, respectively. The notation $\mathbf{X}_k$ is also used to denote the $k^{th}$ frontal slice of a tensor. These are displayed in Figure 2.3.

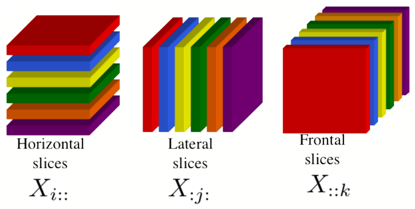


FIG. 2.3. *Slices of a third order tensor.*

In addition, there are tensor symbols we need to define. The symbol $\circ$ denotes the outer product of vectors. For example, for $\mathbf{a} \in \mathbb{R}^I$, $\mathbf{b} \in \mathbb{R}^J$ and $\mathbf{c} \in \mathbb{R}^K$, we can form a tensor $\mathcal{X}$ from the outer product, $\mathcal{X} = \mathbf{a} \circ \mathbf{b} \circ \mathbf{c}$ where $x_{ijk} = a_i b_j c_k$ for all $i = 1, \ldots, I$, $j = 1, \ldots, J$, and $k = 1, \ldots, K$. We define the Hadamard (i.e., elementwise) matrix product using the symbol $*$. The Khatri-Rao product [18, 22, 2, 24] is a columnwise Kronecker product. For two matrices $\mathbf{A} \in \mathbb{R}^{I \times P}$ and $\mathbf{B} \in \mathbb{R}^{J \times P}$, the Khatri-Rao product is

$$\mathbf{A} \odot \mathbf{B} = \begin{bmatrix} \mathbf{a}_1 \otimes \mathbf{b}_1 & \mathbf{a}_2 \otimes \mathbf{b}_2 & \cdots & \mathbf{a}_P \otimes \mathbf{b}_P \end{bmatrix},$$

where $\mathbf{a} \otimes \mathbf{b}$ denotes the matrix Kronecker product.

For some computations, it is necessary to treat the entire tensor in matrix form. To do this we go through a process called matricization. We define $\mathbf{X}_{(1)}$ to the be the mode-1 matricization of $\mathcal{X}$. This means the mode-1 fibers (the column fibers) are aligned to form a matrix. Specifically, the mode-1 fibers are mapped to the rows of a matrix, and the modes-2 and -3 fibers are mapped to the columns of the matrix. See Figure 2.4 for an illustration of mode-1 matricization. We also look at $\mathbf{X}_{(2)}$ and $\mathbf{X}_{(3)}$, the mode-2 (row) and mode-3 (tube) matricizations, respectively. The ordering of the grouped modes makes a difference. Although we do not indicate it explicitly, we assume $\mathbf{X}_{(1)} = \mathbf{X}_{(\{1\}\times\{2,3\})}$, $\mathbf{X}_{(2)} = \mathbf{X}_{(\{2\}\times\{1,3\})}$, and $\mathbf{X}_{(3)} = \mathbf{X}_{(\{3\}\times\{1,2\})}$, per the notation of [14]. This ordering is not universal; see, e.g., [5, 13].

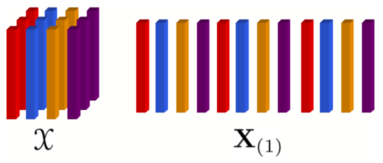


FIG. 2.4. *Example of a mode-1 matricization,* $\mathbf{X}_{(1)}$*. The column fibers are aligned to form a matrix, with the mode-1 fibers as rows in our matrix, and the modes-2 and -3 fibers as the columns of the matrix.*

The tensor norm is defined as the square root of the sum of the squares of all the elements of a tensor. For $\mathcal{X} \in \mathbb{R}^{I\times J\times K}$,

$$\|\mathcal{X}\|^2 = \sum_{i=1}^{I}\sum_{j=1}^{J}\sum_{k=1}^{K} x_{ijk}^2.$$

This is the higher-order analogue of the matrix Frobenius norm.

For more information on tensor notation, see [13, 9, 1, 14].

## 3. Special types of tensors.

We will discuss two special types of tensors in this paper: Kruskal tensors and sp3way tensors.

### 3.1. Kruskal tensors.

Kruskal tensors, named for Kruskal [15, 16], are tensors that are stored as the sum of $R$ rank-1 tensors. If $\mathcal{X} \in \mathbb{R}^{I\times J\times K}$ is a Kruskal tensor, it is stored as:

$$\mathcal{X} = \sum_{r=1}^{R} \mathbf{a}_r \circ \mathbf{b}_r \circ \mathbf{c}_r,$$

where $\mathbf{A} \in \mathbb{R}^{I\times R}$, $\mathbf{B} \in \mathbb{R}^{J\times R}$, and $\mathbf{C} \in \mathbb{R}^{K\times R}$. The notation $\mathbf{a}_r$ denotes the $r^{th}$ column of a matrix $\mathbf{A}$. An illustration of this idea is in Figure 3.1. This type of tensor results from a CANDECOMP/PARAFAC decomposition [3, 7], described in section 4. We use the shorthand notation from [14]:

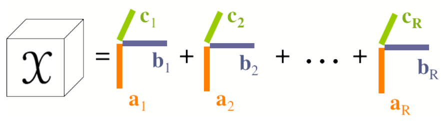


FIG. 3.1. *A third-order Kruskal tensor* $\mathcal{X}$ *is written as the sum of R rank-1 tensors.*

$$\mathcal{X} = [\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!],$$

though other notation can be used. For instance, Kruskal [15] uses

$$\mathcal{X} = (\mathbf{A}, \mathbf{B}, \mathbf{C})\,.$$

It is also possible to have explicit weights $\lambda \in \mathbb{R}^R$. Then

$$\mathcal{X} = \sum_{r=1}^{R} \lambda_r\, \mathbf{a}_r \circ \mathbf{b}_r \circ \mathbf{c}_r, \qquad \text{and} \qquad \mathcal{X} = [\![\lambda; \mathbf{A}, \mathbf{B}, \mathbf{C}]\!].$$

**3.2. A new class of tensors: sp3way tensors.** We have created a new class of tensors that are third-order (3-way) with special structure. For this type of tensor, each slice is dense, but formed from the product of a sparse matrix and its transpose. Specifically, each (frontal) slice of $\mathcal{X} \in \mathbb{R}^{N\times N\times P}$ is written, $\mathbf{X}_p = \mathbf{Y}_p\mathbf{Y}_p^T$ for sparse $\mathbf{Y}_p$, with $p = 1, \ldots, P$. This is shown in Figure 3.2.

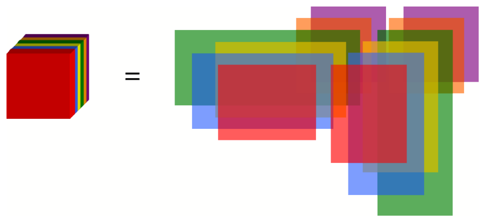


FIG. 3.2. *An sp3way tensor, in which each slice of the tensor, $\mathcal{X}$, is formed from the product of a sparse matrix and its transpose, $\mathbf{X}_p = \mathbf{Y}_p\mathbf{Y}_p^T$ for sparse $\mathbf{Y}_p$, with $p = 1, \ldots, P$.*

The motivation for this type of tensor came from wanting to store multiple similarity matrices simultaneously. Our idea is to view the similarity (object-object) matrices as formed from the product of an object-feature matrix and its transpose. For the tensor $\mathcal{X} \in \mathbb{R}^{N\times N\times P}$ we have $N$ objects and $P$ features. In general, object-feature matrices are sparse. The slices of our final tensor are object-object matrices which can be dense, and each slice is a different similarity matrix. When we do computations on sp3way tensors, everything is done on the sparse matrices directly.

**4. CANDECOMP/PARAFAC (CP) and INDSCAL.** Canonical Decomposition (CANDECOMP) [3] and Parallel Factors (PARAFAC) [7] are two different names for the same tensor decomposition, first published in 1970, and often abbreviated CP so as to give credit to both names. This decomposition is a higher-order analogue of the matrix Singular Value Decomposition (SVD) or matrix Principal Component Analysis (PCA). For a general third-order tensor, a rank-$R$ CP decomposition approximates a tensor $\mathcal{X} \in \mathbb{R}^{I\times J\times K}$ by a Kruskal tensor $\mathcal{K}$. Since $\mathcal{X}$ is a 3-way tensor, the Kruskal tensor is written $\mathcal{K} = [\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!]$.

The standard algorithm for computing a CP decomposition employs Alternating Least Squares (ALS) [3]. The CP-ALS algorithm requires two matrices to begin, say $\mathbf{A}$ and $\mathbf{B}$. We then iterate through three steps to compute the matrices $\mathbf{C}$, $\mathbf{B}$, and $\mathbf{A}$ which form the Kruskal tensor. The algorithm proceeds as follows.

1. Holding $\mathbf{A}$ and $\mathbf{B}$ constant, solve for $\mathbf{C}$:

$$\mathbf{C} = \mathbf{X}_{(3)}\,(\mathbf{B} \odot \mathbf{A})\left(\mathbf{B}^T\mathbf{B} * \mathbf{A}^T\mathbf{A}\right)^{\dagger}.$$

2. Holding $\mathbf{A}$ and $\mathbf{C}$ constant, solve for $\mathbf{B}$:

$$\mathbf{B} = \mathbf{X}_{(2)}\,(\mathbf{C} \odot \mathbf{A})\left(\mathbf{C}^T\mathbf{C} * \mathbf{A}^T\mathbf{A}\right)^{\dagger}.$$

3. Holding **B** and **C** constant, solve for **A**:

$$\mathbf{A} = \mathbf{X}_{(1)} (\mathbf{C} \odot \mathbf{B}) \left( \mathbf{C}^T \mathbf{C} * \mathbf{B}^T \mathbf{B} \right)^{\dagger} .$$

We iterate through the three steps, computing new values for $\mathbf{C}, \mathbf{B}$, and $\mathbf{A}$, until the fit of the Kruskal tensor to the original tensor ceases to improve or we reach our maximum number of allowed iterations.

**4.1. Symmetric CP.** In the original paper by Carroll and Chang [3], in addition to introducing the CANDECOMP decomposition of a tensor, they also introduce individual differences in scaling (INDSCAL), which is a symmetric CP decomposition. To employ INDSCAL, the frontal slices of the tensor $\mathcal{X} \in \mathbb{R}^{N \times N \times P}$ must be symmetric. The major difference is that, unlike normal (not necessarily symmetric) CP-ALS iterations, there is no constraint to make $\mathbf{A} = \mathbf{B}$. However, when we have symmetry in the frontal slices ($x_{ijk} = x_{jik}$ for all $i, j, k$), we are guaranteed to have **A** and **B** equal, or at least related by a diagonal transformation, **D**, by the time the algorithm has converged [3]. Specifically,

$$\mathbf{A} = \mathbf{D}\mathbf{B}$$
$$\mathbf{B} = \mathbf{D}^{-1}\mathbf{A}$$

where $\mathbf{D} \in \mathbb{R}^{N \times N}$ is a diagonal matrix with nonzero diagonal elements.

Two alternatives to the CP-ALS algorithm exist to explicitly obtain $\mathbf{A} = \mathbf{B}$, even if the frontal slices of the tensor are not symmetric [3]. The first option is not recommended [3] since its properties are not well understood. In this option, the third step becomes:

3. Set $\mathbf{A} = \mathbf{B}$.

Then, as before, we iterate through steps 1, 2, and 3. A better alternative is to follow the initially laid out steps until the tolerance is reached. Then set $\mathbf{A} = \mathbf{B}$, and solve for **C** one final time.

**4.2. Uniqueness Conditions for CP.** There is a long history on uniqueness conditions for decompositions, and Stegman, et al. [25] give a good summary. The first uniqueness results on CP are credited to Jennrich (in the Acknowledgments section by Harshman [7], 1970) and Harshman [8], 1972. The most general sufficient condition for uniqueness is due to Kruskal [15] in 1977. Specifically, Kruskal defined the $k$-rank of a matrix, which is the largest number $k$ such that every subset of $k$ columns of the matrix is linearly independent. We denote the $k$-rank of a matrix **A** as $k_{\mathbf{A}}$. Kruskal's condition states:

$$k_{\mathbf{A}} + k_{\mathbf{B}} + k_{\mathbf{C}} \geq 2R + 2$$

is sufficient for uniqueness of CP, where $\mathcal{X} \in \mathbb{R}^{I \times J \times K}$, written $\mathcal{X} = [\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!]$, is rank $R$.

Some additional uniqueness conditions have been developed more recently. In 2000, Sidiropoulos and Bro [23] illustrated a short-cut proof for the uniqueness conditions and generalized the result to n-way tensors (for $n > 3$). In 2002, ten Berge and Sidiropoulos [26] proved that Kruskal's sufficiency condition is also necessary for $R = 2$ and $R = 3$, but not for $R > 3$, (uniqueness for $R = 1$ was proved by Harshman [8]). Some alternate uniqueness conditions are given by Jiang and Sidiropoulos, and De Lathauwer in [12, 4]. These two references separately examine the case where one of the component matrices is of full column rank. Each paper assumes three conditions (though not the same conditions). Stegman, et al. [25], take the same approach as in [4], however [25] should be more accessible since there are no fourth order tensors and the only requirement is basic linear algebra. In addition, Stegman et al. [25] state distinct general uniqueness conditions for CP and INDSCAL. For

INDSCAL, stricter uniqueness conditions in terms of $R$ are obtained because the model has less parameters. The conditions for CP are related to those from [12, 4].

The major idea from Stegman, et al. [25] is that we almost surely have uniqueness for the CP and INDSCAL decompositions when $\mathbf{A}$ and $\mathbf{B}$ are randomly sampled from a continuous distribution and $\mathbf{C}$ has full rank. Further, in a majority of situations, the CP and INDSCAL algorithms can be thought of as randomly sampled from a continuous distribution. Thus, the uniqueness conditions should, but do not always, apply.

**5. Implicit Slice Canonical Decomposition (IMSCAND).** The motivation for a new decomposition comes from the new sp3way tensors, which allow us to work with a dense tensor, but only requires us to store sparse matrices. We call this new decomposition IMSCAND for Implicit Slice Canonical Decomposition. This decomposition gives a canonical decomposition of a tensor whose slices are formed implicitly as the product of a sparse matrix and its transpose.

Both CP and IMSCAND produce the same decomposition of a tensor into a Kruskal tensor using the same number of iterations. The decompositions have a major difference however. IMSCAND stores sparse matrices, $\mathbf{Y}_i$, which are implicitly multiplied by their transpose to form the frontal slices of the tensor. Specifically, the frontal slices of $\mathcal{X} \in \mathbb{R}^{N\times N\times P}$ are

$$\mathbf{X}_p = \mathbf{Y}_p\mathbf{Y}_p^T \text{ for } p = 1,\dots,P.$$

All computations are done on the sparse matrices directly. CP, on the other hand, stores fully formed slices, which can be dense. Because all the computations are done for sparse matrices, IMSCAND is capable of handling much larger problems than CP in less time.

The algorithm for computing IMSCAND differs from the CP-ALS algorithm at three main points. The difference occurs in the calculations of $\mathbf{X}_{(3)}(\mathbf{B}\odot\mathbf{A})$, $\mathbf{X}_{(2)}(\mathbf{C}\odot\mathbf{A})$, and $\mathbf{X}_{(1)}(\mathbf{C}\odot\mathbf{B})$. The CP-ALS algorithm follows:

1. Holding $\mathbf{A}$ and $\mathbf{B}$ constant, solve for
$$\mathbf{C} = \widehat{\mathbf{C}}\left(\mathbf{B}^T\mathbf{B} * \mathbf{A}^T\mathbf{A}\right)^{\dagger},\ \text{with } \widehat{\mathbf{C}} = \mathbf{X}_{(3)}(\mathbf{B}\odot\mathbf{A}).$$
2. Holding $\mathbf{A}$ and $\mathbf{C}$ constant, solve for
$$\mathbf{B} = \widehat{\mathbf{B}}\left(\mathbf{C}^T\mathbf{C} * \mathbf{A}^T\mathbf{A}\right)^{\dagger},\ \text{with } \widehat{\mathbf{B}} = \mathbf{X}_{(2)}(\mathbf{C}\odot\mathbf{A}).$$
3. Holding $\mathbf{B}$ and $\mathbf{C}$ constant, solve for
$$\mathbf{A} = \widehat{\mathbf{A}}\left(\mathbf{C}^T\mathbf{C} * \mathbf{B}^T\mathbf{B}\right)^{\dagger}\ \text{with } \widehat{\mathbf{A}} = \mathbf{X}_{(1)}(\mathbf{C}\odot\mathbf{B}).$$

**5.1. Computing the product of a tensor and 2 vectors in 2 modes.** The entire point of the IMSCAND decomposition is to not form any of the $\mathbf{X}_i$ directly. Thus we have different ways to compute $\widehat{\mathbf{C}}$, $\widehat{\mathbf{B}}$ and $\widehat{\mathbf{A}}$. These are all computed using the sparse $\mathbf{Y}_i$ object-feature matrices directly.

We compute $\widehat{\mathbf{C}} \in \mathbb{R}^{P\times R}$ elementwise using $\mathbf{A}$ and $\mathbf{B}$. Define $\mathbf{b}_r$ as the $r^{th}$ column of $\mathbf{B}$, and $\mathbf{a}_r$ as the $r^{th}$ column of $\mathbf{A}$. Then the $(p, r)$ element of $\widehat{\mathbf{C}}$ is

$$\widehat{c}_{pr} = \left(\mathbf{Y}_p^T\mathbf{b}_r\right)^T\left(\mathbf{Y}_p^T\mathbf{a}_r\right),\ \text{for } p = 1,\dots,P \text{ and } r = 1,\dots,R.$$

Both $P$ and $R$ are relatively small.

The values for $\widehat{\mathbf{B}}$ and $\widehat{\mathbf{A}}$ are computed with the same formula. Unlike $\widehat{\mathbf{C}}$ which is computed elementwise, these are both computed columnwise. We assume there are $P$ different features and thus $P$ slices in our tensor. Then, using $\mathbf{C}$, $\mathbf{A}$, and notation from $\widehat{\mathbf{C}}$,

$$\widehat{\mathbf{b}}_r = \sum_{p=1}^{P} c_{pr} \left[ \mathbf{Y}_p \left( \mathbf{Y}_p^T \mathbf{a}_r \right) \right].$$

And using $\mathbf{B}$ and $\mathbf{C}$,

$$\widehat{\mathbf{a}}_r = \sum_{p=1}^{P} c_{pr} \left[ \mathbf{Y}_p \left( \mathbf{Y}_p^T \mathbf{b}_r \right) \right].$$

**6. Numerical Results.** We have analyzed both real and simulated data. Our main goal is to gain intuition for the optimal choice of the IMSCAND rank and to increase our understanding of the $k$-means clustering algorithm.

Our data sets are too large and dense for previous tensor decompositions. Thus we have established the new IMSCAND decomposition that exploits the structure of the data to make the necessary computations possible. Specifically, instead of storing the slices, which are dense object-object similarity matrices, we instead store the sparse object-feature matrices which are multiplied by their transpose to form the slices. From the IMSCAND decomposition, we obtain matrices $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{C}$. We start with symmetric data so that our output has the property $\mathbf{A} \approx \mathbf{B}$ (we only get equality when CP converges). This is important, since the rows of both of these matrices contain feature vectors, which are a compilation of all of the different similarities. We want the same feature vectors since $k-$means clustering is used on these vectors to obtain a clustering of our data. We use the matrix $\mathbf{A}$ to obtain our feature vectors.

**6.1. Randomly Generated Data.** Our simulated data was created to produce an sp3way tensor with pre-determined clusters. Specifically, the user determines the number of nodes, number of different similarities (= number of frontal slices), number of clusters, number of objects in each cluster, number of features in each cluster for each different type of feature, and the probabilities that two nodes are in the same cluster or in different clusters for each type of feature.

For example, consider the arranged $5 \times 5 \times 2$ sp3way tensor, $\mathcal{X}$ with 3 clusters. The tensor is formed from object-feature matrices $\mathbf{Y}_1$ and $\mathbf{Y}_2$, with

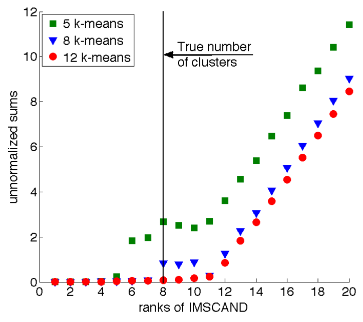
$$\mathbf{Y}_1 = \left( \begin{array}{c|cc|c} * & 0 & 0 & 0 \\ * & 0 & 0 & 0 \\ \hline 0 & * & * & 0 \\ \hline 0 & 0 & 0 & * \\ 0 & 0 & 0 & * \end{array} \right) \qquad \mathbf{Y}_2 = \left( \begin{array}{c|ccc|cc} * & 0 & 0 & 0 & 0 & 0 \\ * & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & * & * & * & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & * & * \\ 0 & 0 & 0 & 0 & * & * \end{array} \right).$$

For both of these matrices, the stars are in-cluster values, and the zeros are out-of-cluster values. The out of cluster values do not have to be zero, and the in-cluster values can be zero. The code is designed so the the clusters are denser than the non-cluster areas. After creating the slices, we rearrange the rows to make the clusters less obvious and the algorithm nontrivial.

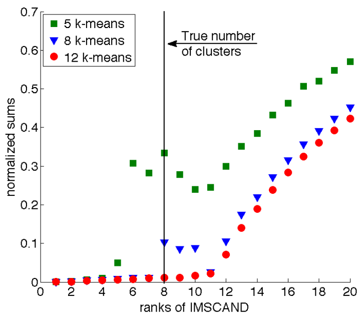
Both object-feature matrices illustrate that there are two objects in the first cluster, one in the second cluster, and two in the third cluster. For the first similarity, $\mathbf{Y}_1$, we see that there is one feature in the first cluster, two features in the second cluster, and one feature in the third cluster. For the second similarity, $\mathbf{Y}_2$ we see that there is one feature in the first cluster, three features in the second cluster, and two features in the third cluster.

As an numerical example, we have generated an sp3way tensor $\mathcal{X} \in \mathbb{R}^{1000\times1000\times6}$ with 8 clusters. The sizes for $\mathbf{Y}_i$ range for the number of features goes from 678 columns to 21342 columns. The number of objects per cluster ranges from 76 to 195, and the number of features per cluster ranges from 8% to 18% of the total number of features for that similarity. The in-cluster probabilities range from 9% to 37%, and the out-of-cluster probabilities range from 0.4% to 21%.

We considered many outputs in hopes that something would be a good indicator of an acceptable choice for the IMSCAND rank. The fit value from IMSCAND does not indicate the correct number of clusters, or the number of clusters requested from $k$-means. We also plotted 2 values from the $k$-means algorithm. The first graph in Figure 6.1(a) is the total sum of the distance between each point and its centroid. The second graph in Figure 6.1(b) is the same sum of distances, only we divide by the rank of IMSCAND or number of dimensions. The second graph is essentially a normalization of the first graph on the number of dimensions. For each graph, we computed multiple IMSCAND decompositions with ranks ranging from 1 to 20. We did three separate $k$-means clusterings, asking for the correct number of clusters (8), more (12), and less (5). In all of these, we can see a jump in the graph at the $k$-means clustering value, but not at the actual number of clusters.



(a) Unnormalized Sums



(b) Normalized Sums

FIG. 6.1. *Both graphs look at the total sum of the distances from each points to its k-means centroid. The normalized sums are divided by the number of dimensions (which also corresponds to the rank of IMSCAND.*

**6.2. The SIAM Journal Data.** The data set we are using with IMSCAND is a set of approximately 4700 articles from eleven Society of Industrial and Applied Mathematics (SIAM) journals and SIAM proceedings (SIAM PROC S) for a five-year period from 1999 to 2004. The names of the publications used throughout this paper are the ISI abbreviations[1] for the journals. We will use the terms *article* and *document* interchangeably.

There are both explicit links and implicit links built in to the journal data. An explicit link exists when one paper cites another paper. The implicit links include connections between papers through similar authors, title words, abstract words and author-specified keywords.

For this application, we have a tensor with six frontal slices, each of which is formed

[1]`http://www.isiknowledge.com/`

from the product of a sparse article-feature matrix and its transpose. The slices are:

$$\begin{aligned}
\mathbf{X}_1 &= \text{similarity between words in the abstract}\\
\mathbf{X}_2 &= \text{similarity between names of authors}\\
\mathbf{X}_3 &= \text{similarity between author-specified keywords}\\
\mathbf{X}_4 &= \text{similarity between words in the title}\\
\mathbf{X}_5 &= \text{co-citation information}\\
\mathbf{X}_6 &= \text{co-reference information}
\end{aligned}$$

We discuss these in more detail. The first four slices are formed as $\mathbf{X}_i = \mathbf{Y}_i\mathbf{Y}_i^T$ for $i = 1, \ldots, 4$.

1. Let $\mathbf{Y}_1$ be the adjacency matrix formed using information about the abstracts of the papers. Specifically, we form a document-term matrix where the words in the abstracts are used to determine the terms. The documents are listed in the rows, and there is a word bank for the abstracts that is used to form the columns. The word bank includes all word appearing in abstracts, except for common "stop-list" words. There are more than 8000 columns in $\mathbf{Y}_1$.
2. Let $\mathbf{Y}_2$ be the adjacency matrix formed using the documents as rows and the authors as columns. There are more than 6000 authors associated with the data.
3. Let $\mathbf{Y}_3$ be the adjacency matrix formed using the documents as rows and authors-specified keywords as columns. There are less keywords than documents (just over 3000).
4. Let $\mathbf{Y}_4$ be the adjacency matrix formed using the documents as rows and a word bank of terms from the titles as columns. There are less than 3000 distinct words that appear in the titles.
5. For $\mathbf{X}_5$ (and $\mathbf{X}_6$), we need the sparse citation matrix, $\mathbf{C}$. The citation matrix is a binary matrix defined such that
   $$c_{ij} = \begin{cases} 1, & \text{if paper } i \text{ cites paper } j \\ 0, & \text{otherwise} \end{cases}.$$
   Then the co-citation matrix is defined $\mathbf{X}_5 \equiv \mathbf{C}^T\mathbf{C}$ with
   $$(\mathbf{X}_5)_{ij} = \text{ the number of papers citing both documents } i \text{ and } j.$$
6. For $\mathbf{X}_6$, as with $\mathbf{X}_5$, the sparse citation matrix, $\mathbf{C}$ is used (see 5. above). The co-reference matrix is defined $\mathbf{X}_6 \equiv \mathbf{C}\mathbf{C}^T$ with
   $$(\mathbf{X}_6)_{ij} = \text{ the number of papers cited by both documents } i \text{ and } j.$$

Although the original dataset contained 11 journals and SIAM proceedings, after initial trials we decided to remove two of the journals and SIAM proceedings. We removed the SIAM Journal on Applied Dynamical Systems because it had only 32 eligible papers in the five year span. We did not believe there would be enough information from these papers to be able to identify these documents in the clustering. We also removed SIAM Review and SIAM proceedings since they had documents spanning a variety of different topics, but lacked one unifying area by which they might later be clustered together.

Before running IMSCAND and $k$-means, we first normalized the slices by dividing each row of each slice by its 2-norm. The first four slices have naturally higher overall weights (measured by the Frobenius norm), due to more links in the matrices.

To give an example of some results, we computed an IMSCAND rank-20 decomposition on the nine SIAM journals. The confusion matrix follows with the clusters from $k$-means as the columns, and the actual journals in which the papers were published as rows:

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| SIAM J Appl Math | 390 | 114 | 13 | 0 | 19 | 0 | 1 | 0 | 8 |
| SIAM J Comput | 0 | 377 | 11 | 142 | 0 | 4 | 0 | 4 | 0 |
| SIAM J Control Optim | 57 | 311 | 171 | 1 | 3 | 5 | 4 | 16 | 8 |
| SIAM J Discrete Math | 0 | 126 | 2 | 128 | 0 | 1 | 1 | 1 | 0 |
| SIAM J Math Anal | 234 | 115 | 4 | 0 | 41 | 3 | 2 | 0 | 18 |
| SIAM J Matrix Anal A | 12 | 135 | 2 | 7 | 1 | 255 | 3 | 6 | 1 |
| SIAM J Numer Anal | 98 | 176 | 7 | 1 | 59 | 26 | 197 | 3 | 44 |
| SIAM J Opitmiz | 0 | 173 | 12 | 8 | 0 | 8 | 1 | 142 | 0 |
| SIAM J Sci Comput | 85 | 244 | 7 | 5 | 37 | 118 | 103 | 7 | 48 |

An associated 3-D bar chart is illustrated in Figure 6.2.

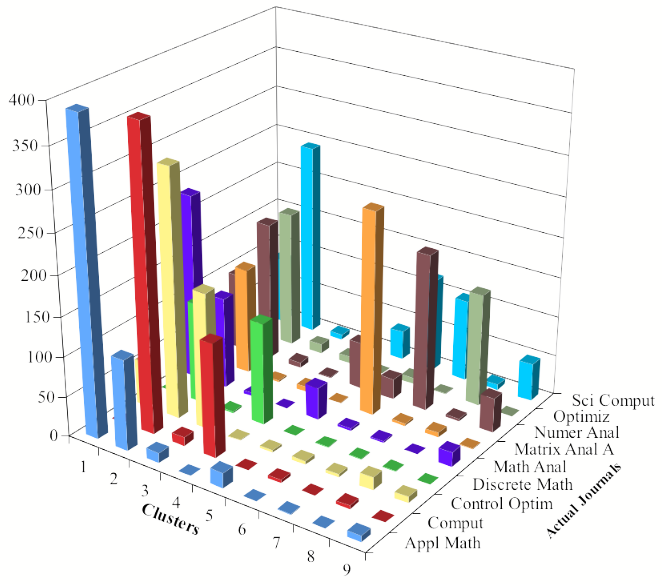


FIG. 6.2. *This chart contains the same information as the confusion matrix. The papers come from 9 SIAM journals, and we have asked k-means to find 9 clusters. The height of each column is the number of documents.*

We can see by looking at the columns of the confusion matrix that two of the clusters, #3 and #8, find papers mostly from one journal. However, if we look at these specific journals, these clusters are not getting all the papers for the journal, just a large portion of them. Looking at clusters (columns) #1, #4, #6, and #7 we are mostly getting papers from two journals. We can easily make an argument that there are documents with related authors; title, abstract, and key words; and citations in each of these different pairs of journal, and thus it is logical that we have clusters containing documents from two journals. Additionally, the SIAM J Sci Comput is a rather diverse journal, and thus occurs in many different predicted journals.

We have more than forty percent of the papers being clusters together in cluster #2. We do not have an explanation for this occurrence. It may be that the relatively sparse co-citation and co-reference slices have some impact on this. When IMSCAND and $k$-means were run only on these two slices, more than 90% of the documents were clustered together. However, IMSCAND and $k$-means on the other four slices only (abstract, author, keyword, and title) still yields a cluster containing almost 50% of the data.

**7. Future Work: Clustering Attributed Relational Graphs for Information Organization (CARGIO).** CARGIO is a multi-year project whose goal is to determine clusters from a set of files on a computer hard drive. Images of several hard drives are currently being examined in the groundtruthing process, to determine how the files are clustered. This groundtruth test data will be used to help identify the correct procedure for future hard drives where the clustering is unknown. For more information, see Appendix A.

**8. Conclusions.** We addressed the problem of clustering objects from datasets with multiple similarity measures. Our approach includes the creation of a new class of tensors, sp3way tensors, which allow the dense similarity matrices to be stored implicitly as the product of a sparse object-feature matrix and its transpose. In addition, we have created a new tensor decomposition, IMSCAND, which is identical in output to CP, but is computed on the sparse object-feature matrices only. This new decomposition allows us to handle much larger problems than if we stored the full similarity matrices. We have computed IMSCAND on both generated and real data and tried to form some conclusions on the optimal ranks of IMSCAND with regard to identifying the best number of clusters to represent a group of data.

## REFERENCES

[1] B. W. Bader and T. G. Kolda, *Algorithm 862: MATLAB tensor classes for fast algorithm prototyping*, ACM Transactions on Mathematical Software, 32 (2006).

[2] R. Bro, *Multi-way analysis in the food industry: Models, algorithms, and applications*, PhD thesis, University of Amsterdam, 1998.

[3] J. D. Carroll and J.-J. Chang, *Analysis of individual differences in multidimensional scaling via an $N$−way generalization of "Eckart-Young" decomposition*, Psychometrika, 35 (1970), pp. 283–319.

[4] L. De Lathauwer, *A link between the canonical decomposition in multilinear algebra and simultaneous matrix diagonalization*, SIAM Journal for Matrix Analysis and its Applications, 28 (2006), pp. 642–666.

[5] L. De Lathauwer, B. De Moor, and J. Vandewalle, *A multilinear singular value decomposition*, SIAM Journal for Matrix Analysis and its Applications, 21 (2000), pp. 1253–1278.

[6] D. M. Dunlavy, T. G. Kolda, and W. P. Kegelmeyer, *Multilinear algebra for analyzing data with multiple linkages*, Tech. Report SAND2006-2079, Sandia National Laboratories, Albuquerque, NM and Livermore, CA, 2006.

[7] R. A. Harshman, *Foundations of the PARAFAC procedure: Models and conditions for an "explanatory" multimodal factor analysis*, UCLA Working Papers in Phonetics, 16 (1970), pp. 1–84.

[8] ———, *Determination and proof of minimum uniqueness conditions for PARAFAC-1*, UCLA Working Papers in Phonetics, 22 (1972), pp. 111–117.

[9] ———, *An index formulism that generalizes the capabilities of matrix notation and algebra to n-way arrays*, J. Chemometr., 15 (2001), pp. 689–714.

[10] M. A. Jaro, *Advances in record linking methodology as applied to the 1985 census of Tampa, Florida*, Journal of the American Statistical Society, 64 (1989), pp. 1183–1210.

[11] ———, *Probabilistic linkage of large public health data file*, Statistics in Medicine, 14 (1995), pp. 491–498.

[12] T. Jiang and N. D. Sidiropoulos, *Kruskal's permutation lemma and the identification of CANDECOMP/-PARAFAC and bilinear models with constant modulus constraints*, IEEE Transactions on Signal Processing, 52 (2004), pp. 2625–2636.

[13] H. A. L. Kiers, *Towards a standardized notation and terminology in multiway analysis*, J. Chemometr., 14 (2000), pp. 105–122.

[14] T. G. Kolda, *Multilinear operators for higher-order decompositions*, Tech. Report SAND2006-2081, Sandia National Laboratories, Albuquerque, NM and Livermore, CA, 2006.

[15] J. B. Kruskal, *Three-way arrays: Rank and uniqueness of trilinear decompositions, with applications to arithmetic complexity and statistics*, Linear Algebra and its Applications, 18 (1977), pp. 95–138.

[16] ———, *Rank, decomposition, and uniqueness for 3-way and n-way arrays*, in Multiway Data Analysis, Elsevier Science Publishers B.V., 1989, pp. 7–18.

[17] H. W. Kuhn, *The Hungarian method for the assignment problem*, Naval Research Logistics Quarterly, 2 (1955), pp. 83–87.

[18] R. P. McDonald, *A simple comprehensive model for the analysis of covariance structures*, Brit J. Math. Stat. Psy., 33 (1980), p. 161.

[19] M. Meilă, *Comparing clusterings*, Tech. Report 418, University of Washington, Dept. of Statistics, 2002.

[20] ———, *Comparing clustering by the variation of information*, in Lecture notes in Computer Science, B. Schölkopf and M. K. Warmuth, eds., vol. 2777, Springer, 2003, pp. 173–187.
[21] ———, *Comparing clustering – an axiomatic view*, in Proceedings of the 22nd International Conference on Machine Learning, 2005, pp. 577–584.
[22] C. R. Rao and S. Mitra, *Generalized inverse of matrices and its applications*, Wiley, New York, 1971.
[23] N. D. Sidiropoulos and R. Bro, *On the uniqueness of multilinear decompositions of n-way arrays*, Journal of Chemometrics, 14 (2000), pp. 229–239.
[24] A. Smilde, R. Bro, and P. Geladi, *Multi-way analysis: Applications in the chemical sciences*, Wiley, 2004.
[25] A. Stegeman, J. M. F. ten Berge, and L. D. Lathauwer, *Sufficient conditions for uniqueness in CANDECOMP/PARAFAC and INDSCAL with random component matrices*, Psychometrika, 71 (2006), pp. 219–229.
[26] J. M. F. ten Berge and N. D. Sidiropoulos, *On uniqueness in CANDECOMP/PARAFAC*, Psychometrika, 67 (2002), pp. 399–409.
[27] S. van Dongen, *Performance criteria for graph clustering and Markov cluster experiments*, Tech. Report INS-R0012, Center for Mathematics and Computer Science (CWI), Amsterdam, The Netherlands, 2000.
[28] W. E. Winkler, *The state of record linkage and current research problems*, Internal Revenue Service Publication R99/04, 1999.

**Appendix A. Clustering Attributed Relational Graphs for Information Organization (CARGIO).** We are currently considering seven different attributes of the files, each of which produces an adjacency matrix which becomes one slice of a tensor. Three of the edges (*parent, ancestry*, and *symbolic link*) are directed, and the other four (*time delta, sibling, name match*, and *text match*) are undirected.

- *time delta:* This data yields a completely dense adjacency matrix. Specifically, for any set of two nodes, the difference in seconds between time stamps is used to compute the weight. If $d_{ij}$ is the difference in seconds, then the weight is $1/(1 + d_{ij})$.
- *parent:* This is binary data. If file $i$ is contained within folder $j$, then the $(i, j)$ entry of the adjacency matrix is 1. Otherwise, it has value 0.
- *sibling:* Files $i$ and $j$ are siblings if they are contained inside the same directory. The integer edge weight between them gives their depth in the directory.
- *ancestry:* Perhaps the easiest way to think of this is that the ancestry distance between 2 points is how many of the Unix "cd .." commands need to be executed to move from node $i$ to node $j$. If it is not possible to move from $i$ to $j$ in this way, then the distance is $\infty$. The weight, $w_{ij}$ is computed as $1/(1 + d_{ij})$.
- *name match:* This is a measurement of how much two filenames match. Higher weights are given to files whose prefix letters are more similar than those whose suffix letters are similar. The weighting is done using the Jaro-Winkler distance metric [28] which is a measure of similarity between two strings. This is a variant of the Jaro distance metric [10, 11].
- *text match:* This measures how well the text matches within two files for normal ASCII text files only. For each file, a word bag is created, then compared to other word bags. The similarity measurement being used is the Tanimoto coefficient (Extended Jaccard Coefficient — see `http://en.wikipedia.org/wiki/Jaccard\_index`).
- *symbolic link:* This is currently a zero matrix, or an extremely sparse binary matrix. If $i$ is a symbolic link to $j$, then the $(i, j)$ element is 1.

**A.1. Adjusting CARGIO data for use with IMSCAND.** In its current form, this problem is not computationally possible for the test hard drives. Every sample hard drive has at least 4500 files. A tensor, $\mathcal{X}$ to store this data would be $\in \mathbb{R}^{4500\times4500\times7}$. The slices range from having 3 to more than 21 million nonzero elements. There are more than 40 million nonzero elements total, which becomes too much for an average computer during the CP-ALS algorithm.

We are currently working to adjust the edge types so that they may be treated by the IMSCAND decomposition. We have extended the IMSCAND decomposition for this application to include the ability to treat both sparse, symmetric similarity (object-object) matrices and sparse object-feature matrices (which are multiplied by their transpose to form a tensor slice).

It is not immediately obvious how to adjust *time delta*. One possibility is to form a sparse, binary file-time matrix, where the columns are different timeframes, e.g., 1 minute, 10 minutes, 1 hour, 10 hours, 1 day, 1 week, 1 month, 3 months, 6 months, 1 year, > 1 year. Then if file *i* was altered in the last 7 minutes, there would be a 1 in row *i* in the 10 minute column. The similarity matrix is formed as the product of the file-time matrix and its transpose.

The *parent*, *sibling*, and *ancestry* matrices are all formed from the same starting matrix. Consider the set of 8 nodes with edges illustrated in Figure A.1.

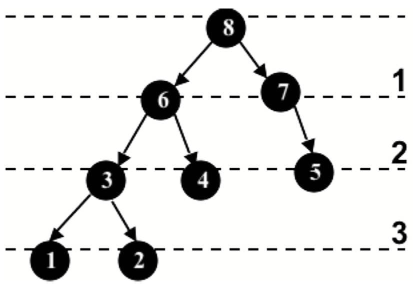


FIG. A.1. *Each node represents a computer file. Node 3 is a parent to nodes 1 and 2. Node 6 is a parent to nodes 3 and 4. Node 7 is a parent to node 5, and node 8 is a parent to nodes 6 and 7.*

We represent this graph as a matrix where the $(i, j)$ element of the matrix is 1 if file $j$ is the parent of file $i$.

$$\mathbf{P} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \end{array} \begin{array}{c} \begin{array}{cccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \end{array} \\ \left(\begin{array}{cccccccc} & & 1 & & & & & \\ & & 1 & & & & & \\ & & & & & 1 & & \\ & & & & & 1 & & \\ & & & & & & 1 & \\ & & & & & & & 1 \\ & & & & & & & 1 \\ & & & & & & & \end{array}\right) \end{array}$$

Technically, the matrix $\mathbf{P}$ is the parent matrix. However, for implementation we need symmetry, thus we could use $\mathbf{P} + \mathbf{P}^T$ to represent the parent-child relationship.

We have two options for the sibling matrix. We could compute $\mathbf{P}\mathbf{P}^T$ directly and get sibling information without weights. The other option is to compute a new matrix $\widehat{\mathbf{P}}$ whose edges are weighted corresponding to the square root of their depth in the tree, then compute $\widehat{\mathbf{P}\mathbf{P}^T}$. With $\widehat{\mathbf{P}}$ defined as:

$$\begin{array}{cccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \end{array}$$

$$\widehat{\mathbf{P}} = \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \end{array} \begin{pmatrix} & & & & \sqrt{3} & & & \\ & & & & \sqrt{3} & & & \\ & & & & & \sqrt{2} & & \\ & & & & & \sqrt{2} & & \\ & & & & & & \sqrt{2} & \\ & & & & & & & 1 \\ & & & & & & & 1 \\ & & & & & & & \end{pmatrix}$$

we compute two options for the *sibling* matrices:

$$\mathbf{P}\mathbf{P}^T = \begin{array}{c|cccccccc} & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ \hline 1 & 1 & 1 & & & & & & \\ 2 & 1 & 1 & & & & & & \\ 3 & & & 1 & 1 & & & & \\ 4 & & & 1 & 1 & & & & \\ 5 & & & & & 1 & & & \\ 6 & & & & & & 1 & 1 & \\ 7 & & & & & & 1 & 1 & \\ 8 & & & & & & & & \end{array}, \quad \widehat{\mathbf{P}\mathbf{P}^T} = \begin{array}{c|cccccccc} & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ \hline 1 & 3 & 3 & & & & & & \\ 2 & 3 & 3 & & & & & & \\ 3 & & & 2 & 2 & & & & \\ 4 & & & 2 & 2 & & & & \\ 5 & & & & & 2 & & & \\ 6 & & & & & & 1 & 1 & \\ 7 & & & & & & 1 & 1 & \\ 8 & & & & & & & & \end{array}$$

Finally, we can get binary ancestry information by first looking at powers of the original matrix $\mathbf{P}$. Matrix $\mathbf{P}^2$ gives grandparent information, $\mathbf{P}^3$ gives great-grandparent information, etc. For the current example, all powers of $\mathbf{P} \geq 4$ are zero.

$$\mathbf{P}^2 = \begin{array}{c|cccccccc} & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ \hline 1 & & & & & & 1 & & \\ 2 & & & & & & 1 & & \\ 3 & & & & & & & & 1 \\ 4 & & & & & & & & 1 \\ 5 & & & & & & & & 1 \\ 6 & & & & & & & & \\ 7 & & & & & & & & \\ 8 & & & & & & & & \end{array} \quad \mathbf{P}^3 = \begin{array}{c|cccccccc} & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ \hline 1 & & & & & & & & 1 \\ 2 & & & & & & & & 1 \\ 3 & & & & & & & & \\ 4 & & & & & & & & \\ 5 & & & & & & & & \\ 6 & & & & & & & & \\ 7 & & & & & & & & \\ 8 & & & & & & & & \end{array}$$

With these powers of the matrix $\mathbf{P}$, we describe how to form the matrix $\mathbf{A}$, which has a sparse structure and can be multiplied by its transpose to form the full *ancestry* similarity matrix. We must first decide how many levels of ancestry are important. A value of 8 may be reasonable, since that would give ancestry information for up to eight levels of files (or up to 7 subfolders). For this example, however, we can choose 3 since all larger powers produce a zero matrix. Then, define the *ancestry* matrix as:

$$\mathbf{A} = \sum_{k=0}^{3} \frac{\mathbf{P}^k}{k!}.$$

Notice this is an approximation of the matrix exponential, $e^A = \sum_{k=0}^{\infty} \mathbf{P}^k/k!$. The matrix $\mathbf{A}$ is our ancestry object-feature matrix, and can be multiplied by its transpose to form a dense ancestry similarity matrix.

Both *name match* and *text match* are computed similarly. For each of these we compute a different list of appropriate terms that are used as the columns of a file-term matrix. This produced a binary matrix in which a value of 1 indicates that the word associated with that column is found in the name (or the text inside) of the file.

The final slice, *symbolic link* is an extremely sparse binary matrix. We create its slice by adding the binary matrix to its transpose in order to obtain a symmetric matrix.

**A.2. Numerical Results.** We have analyzed both real and simulated data. Our main goal is to gain intuition for the choice of the CP rank and to increase our understanding of the $k$-means clustering algorithm. We believe our first attempt at generating data was too simple. We have not yet determined the effectiveness of our second code at generating useful data. Both of these datasets were only run using CP.

**A.2.1. Performance Metrics.** For real or simulated data when we know the true clustering, we can compare the truth to our $k$-means clustering output. The four metrics investigated are the variation of information (VIC), the Scaled Coverage measure (Dongen), classification error (CE), and Mirkin's metric (Mirkin). While discussing these clusters, we will use the notation $C$ and $C'$ to denote two different clusters. As previously defined $N$ is the number of nodes and $K$ is the number of clusters. We now also define $n_k$ to be the number of nodes in cluster $k$ (we also have $k'$, $K'$ and $n_{k'}$ for the cluster $C'$). Finally, we define $n_{kk'}$ to be the number of nodes in both cluster $k$ in $C$ and cluster $k'$ in $C'$.

The variation of information [19, 20, 21] measures the amount of information gained by moving from $C$ to $C'$ and adds that to the information lost by moving from $C'$ to $C$. Specifically, this function is composed of entropy functions for each of the two clusters, $H(C)$ and $H(C')$, and a function of both clusters which gives their mutual information, $I(C, C')$. We compute the VIC metric as

$$d_{VIC} = H(C) + H(C') - 2I(C, C')$$

where

$$H(C) = -\sum_{k=1}^{K} \frac{n_k}{N} \log \frac{n_k}{N} \qquad \text{and} \qquad I(C, C') = \sum_{k=1}^{K} \sum_{k'=1}^{K'} \frac{n_{kk'}}{N} \log \frac{N \cdot n_{kk'}}{n_k \cdot n_{k'}}.$$

The Scaled Coverage measure is also called the van Dongen metric [27]. This metric gives the number of node substitutions needed to convert clustering $C$ to clustering $C'$. Then, per a suggestion from Meilă in [21], the metric is normalized by dividing by $2N$. Specifically, the normalized metric is computed:

$$d_{dongen} = 1 - \frac{1}{2N} \sum_{k=1}^{K} \max_{k'} n_{kk'} - \frac{1}{2N} \sum_{k'=1}^{K'} \max_{k} n_{kk'}.$$

Classification error (CE)[2] is based on the Kuhn-Munkres (Hungarian) algorithm [17, 21]. It is similar to van Dongen's metric, however it is looking for a unique mapping, whereas van Dongen's approach allows multiple clusters to be mapped to one cluster. The metric is defined by

$$d_{CE}(C, C') = 1 - \frac{1}{N} \max_{\sigma} \sum_{k=1}^{K} n_{k,\sigma(k)}.$$

It is assumed that $K < K'$ and we define $\sigma$ to be an injective map from $\{1, \ldots, K\}$ to $\{1, \ldots, K'\}$. Essentially, CE looks at all possible one-to-one comparisons of clusters in $C$ to clusters in $C'$ and chooses the assignment that minimizes the distance between the two clusterings.

[2]Although this algorithm seems to be a good measure of the clustering accuracy, it can take an extremely long time to compute when we are comparing two clustering schemes that are not very similar. For the time being, we are not computing this metric.

The final metric we consider is Mirkin's metric. This metric gives twice the number of points that are in disagreement between any two clusterings $C$ and $C'$. Another thought is that it gives twice the number of point pairs that are in the same cluster in $C$, but in different clusters in $C'$, or vice versa. This metric is rescaled by dividing by $N^2$. We compute this as

$$d_{mirkin}(C, C') = \frac{1}{N^2}\left(\sum_k n_k^2 + \sum_{k'} n_{k'}^2 - 2\sum_k \sum_{k'} n_{kk'}^2\right).$$

**A.2.2. Our initial set of generalized data.** We are generating data with a predetermined clustering. Our goal is to run CP and compute several performance metrics to determine which CP ranks are best at identifying the correct number of clusters. To run the code we must input the number of nodes, $N$ (= number of rows = number of columns), the number of tensor slices, $P$, and the number of clusters, $K$.

Specifically, we generate a vector of $N$ elements that gives us the clustering labels. We then take the first value in *pcvec* (which we will call *pcvec*(1)) and the first value in *pwvec* (called *pwvec*(1)) and create the adjacency matrix that will be the first slice of our tensor. For any two points, if they are in the same cluster, the probability that there is a link between them is in the range $[pcvec(1), 1]$. If there is a link we can either choose for it to have a random weight in that same range, or we can decide to create a binary adjacency matrix and store a weight of 1. For any two points in different clusters, the probability of a link between them is in the range $[0, pwvec(1)]$. Again, this can either have a random link with the weight falling in the specified range, or a weight of 1. After we have completed this adjacency matrix, we move on to the second elements in *pcvec* and *pwvec* to compute the second slice, and continue until we have all the matrices to produce our tensor.

With the tensor produced, we begin the process of computing CP of various ranks and the corresponding clusters and performance metrics. We initialize each CP using the eigenvectors of $\mathbf{X}_{(2)}\mathbf{X}_{(2)}^T$, where $\mathbf{X}_{(2)}$ is the mode-2 matricization of $\mathcal{X}$. For each CP decomposition computed, we call the (built-in with the Statistics package) $k$-means algorithm to run on the rows of the output matrix $\mathbf{A}$. One option we choose in $k$-means clustering is to run the algorithm multiple times and choose the clustering with the smallest normwise error. This should reduce the odds of $k$-means settling on a false minimum. In addition, if it naturally occurs that a cluster is lost in the $k$-means process, we allow the algorithm to simply continue with one less cluster.

Initially, we plotted multiple outputs, in addition to the metrics, in hopes of seeing some strong indicator for the "best" choice for the rank of CP. In addition to the four performance metrics, we also considered fit, or normwise distance between the original tensor and the Kruskal tensor, from the CP algorithm, the number of iterations it took CP to converge, and a measure from $k$-means of the distance of all points to their centroid for each cluster After several trials, we decided to not compute the CE performance metric for each iteration, since it was taking an extremely long time, especially when the clustering from $k$-means was not very close to the true clustering. We also realized that fit and iteration from CP did not indicate that any rank of CP was better than any other. However, the distance measure of points to their centroid from $k$-means turned out to be quite helpful when divided by the rank of CP. The value of the CP rank is the number of dimension, so dividing the sum by this value is essentially a normalization of the sum over the dimensions.

The included results correspond to a $1000 \times 1000 \times 5$ tensors with approximately 300,000 nonzero entries total and ten true clusters. Simulations were run, asking $k$-means for 7, 10, and 14 clusters. I computed CP ranks for 1 to 30, computing $k$-means clustering at each rank.

For all three values of $k$-means, the performance metrics did not indicate that any rank for CP is better than any other. In general, Mirkin's metric was very low (less than 0.36 for

7 *k*-means, and less than 0.16 for the other two *k*-means), which indicates that the *k*-means clusters were good approximations to the true clusters according to that metric. Dongen's metric was also quite low, never ranging above 0.33. The VIC metric can grow quite large when looking at two very different clusters. We did get some values to be quite low (less than 0.2), and it achieved a maximum value at 1.46. The CE took an extremely long time to run, and in our limited experience, we did not find it to be a good indicator of the best CP rank, so we have omitted computing it.

We do have 2 decent indicators of the true number of clusters. These both occur using an output from *k*-means that gives the sum of the distances from each point to its centroid. We consider the sum directly as well as a normalization in which we divide each sum by the number of dimensions (rank of CP). As long as we ask *k*-means to find a number of clusters greater than or equal to the true number, this jump occurs. When *k*-means is asked to find less than the true number of clusters, the jump occurs at the number of clusters asked for by *k*-means.

The major reason we believe this to be too simplistic is that similar results can be generated from just one slice of the tensor.

**A.2.3. Our second set of generalized data.** One of the major goals with this data was that we wanted to ensure that all (or at least more than one) of the slices were needed to obtain all of the information. We generate data with a predetermined clustering and create the adjacency matrices the same way as the original code. The difference is that we can determine, for each slice (or adjacency matrix), how many of the clusters to ignore. When we ignore a cluster, all of its nodes are treated with the out-of-cluster probabilities of occurring. This code also allows us to restrict the number of nonzero elements per slice.

In the trials we ran using this dataset, we were not able to identify an optimal CP rank. Both normalized and unnormalized sums of distances of *k*-means points to their centroids could indicate the number of clusters requested from *k*-means. In addition, the performance metrics were better once we were computing a CP rank greater than the number of clusters requested from *k*-means. These results coincided with the results from the sp3way trials.

# PYTHON OPTIMIZATION MODELING OBJECTS (PYOMO)

NICOLAS L. BENAVIDES[‡], ROBERT D. CARR[§], AND WILLIAM E. HART[¶]

**Abstract.** We describe the Python Optimization Modeling Objects (Pyomo) package. Pyomo is a Python package that can be used to define abstract problems, create concrete problem instances, and solve these instances with standard solvers. Pyomo provides a capability that is commonly associated with algebraic modeling languages like AMPL and GAMS. We introduce Pyomo by contrasting it with the capabilities of AMPL.

## 1. Introduction.

Algebraic Modeling Languages (AMLs) are high-level programming languages for describing and solving mathematical problems, particularly optimization-related problems [9]. AMLs like AIMMS [1], AMPL [2, 8] and GAMS [5] have programming languages with an intuitive mathematical syntax that supports concepts like sparse sets, indices, and algebraic expressions. AMLs provide a mechanism for defining variables and generating constraints with a concise mathematical representation, which is almost essential for real-world problems that can involve thousands of constraints and variables.

An alternative strategy for modeling mathematical problems is to use a standard programming language in conjunction with a software library that uses object-oriented design to support similar mathematical concepts. Although these modeling libraries sacrifice the intuitive mathematical syntax of an AML, they allow the user to leverage the greater flexibility of standard programming languages. For example, modeling libraries like FLOPC++ [4], OPL [6] enable the solution of large, complex problems within a user-defined application.

This paper describes Pyomo, the Python Optimization Modeling Objects (Pyomo) package. Pyomo is a Python package that can be used to define abstract problems, create concrete problem instances, and solve these instances with standard solvers. Like other modeling libraries, Pyomo can generate problem instances and apply optimization solvers with a fully expressive programming language. Further, Python is a noncommercial language with a very large user community, which will ensure robust support for this language on a wide range of compute platforms.

Python is a powerful dynamic programming language that has a very clear, readable syntax and intuitive object orientation. Python's clean syntax allows Pyomo to express mathematical concepts with a reasonably intuitive syntax. Further, Pyomo can be used within an interactive Python shell, thereby allowing a user to interactively interrogate Pyomo-based models. Thus, Pyomo has many of the advantages of both AML interfaces and modeling libraries.

Pyomo makes a clear distinction between the abstract specification of a model, generation of model instances, and the solution of model instances. Abstract models are a key element of AML's like AMPL, and this capability clearly distinguishes Pyomo from other Python modeling libraries like CVXOpt [3] and PuLP [7]. Pyomo models can be solved with either Python optimizers, or with externally defined solvers (e.g. GLPK, CPLEX and CBC). Further, Python can integrate extension modules in low level languages like C or C++ to directly leverage fast solver libraries, and wrapped modules can be used within Python exactly like native Python code.

Section 2 illustrates how Pyomo would be used to model a simple application. We compare and contrast the Pyomo formulation with a formulation developed in the widely used AMPL modeling language. Section 3 describes the Pyomo classes that are used to define

[‡]Santa Clara University, NBenavides@scu.edu
[§]Sandia National Laboratories, rdcarr@sandia.gov
[¶]Sandia National Laboratories, wehart@sandia.gov

model components.

**2. A Simple Example.** In this section we illustrate Pyomo's syntax and capabilities by demonstrating how a simple AMPL example can be replicated with Pyomo Python code.

Consider the basic AMPL program prod.mod:

```
set P;

param a {j in P};
param b;
param c {j in P};
param u {j in P};

var X {j in P};

maximize Total_Profit: sum {j in P} c[j] * X[j];

subject to Time: sum {j in P} (1/a[j]) * X[j] <= b;

subject to Limit {j in P}: 0 <= X[j] <= u[j];
```

To translate this into Pyomo, the user must first import the Pyomo module and create a Pyomo **Model** object:

```
#
# Import Pyomo
#
from pyomo import *

#
# Create model
#
model = Model()
```

This import assumes that Pyomo is available on the users's Python path (see Python documentation for PYTHONPATH for further details). Next, we create the sets and parameters that correspond to the data used in the AMPL model. This can be done very intuitively using the **Set** and **Param** classes.

```
model.P = Set()

model.a = Param(index=model.P)
model.b = Param()
model.c = Param(index=model.P)
model.u = Param(index=model.P)
```

Note that parameter $b$ is a scalar, while parameters $a$, $c$ and $u$ are arrays indexed by the set $P$. Pyomo also defines the **ProductSet** class, which can be defined in a similar manner.

Next, we define the decision variables in this model.

```
def X_bounds(j, model):
    return (0,model.u[j])
model.X = Var(index=model.P, bounds=X_bounds)
```

Decision variables and model parameters are used to define the objectives and constraints in the model. Parameters define constants and the variables are the values that are optimized. Parameter values are typically defined by a data file that is processed by Pyomo.

Objectives and constraints are explicitly defined expressions in Pyomo. The **Objective** and **Constraint** classes require a **rule** option that specifies how these expressions are constructed. This is a function that takes one or more arguments: the first arguments are indices into a set that defines the set of objectives or constraints that are being defined, and the last argument is the model that is used to define the expression.

```
def Objective_rule(model):
    ans = 0
    for j in model.P:
      ans = ans + model.c[j] * model.X[j]
    return ans
model.profit = Objective(rule=Objective_rule)

def Time_rule(model):
    ans = 0
    for j in model.P:
      ans = ans + (1.0/model.a[j]) * model.X[j]
    return ans < model.b
model.Time = Constraint(rule=Time_rule)
```

The rules used to construct these objects use standard Python functions. Finally, note that the **Time_rule** function includes the use of < and > operators on the expression. These operators are used to define upper and lower bounds on the constraints.

Once an abstract model has been created, it can be printed as follows:

```
print ''ABSTRACT MODEL''
model.pprint()
```

This summarize the information in the Pyomo model, but it does not print out explicit expressions. This is due to the fact that an abstract model needs to be instantiated with data to generate the model objectives and constraints:

```
instance = model.create(''prod.dat'')

print ''MODEL INSTANCE''
instance.pprint()
```

Appendix A shows the final Python code for this example.

Once a model instance has been constructed, an optimizer can be applied to it to find an optimal solution. For example, the PICO integer programming solver can be used within Pyomo as follows:

```
opt = solvers.PICO(path="/home/wehart/bin/PICO", keepFiles=True)
solutions = opt.solve(instance)
```

This creates an optimizer object for the PICO executable defined in a given path, and it indicates that temporary files should be kept. The Pyomo model is handed to this optimizer, which returns the final solutions generated by the optimizer.

**3. Documentation of Pyomo Objects.** In this section we provide more detail on the definitions of Pyomo classes that are used to define models.

**3.1. Sets.** The **Set()** class is used to index other objects (e.g. **Param** and **Var**). This class has the same look-and-feel as a **sets.Set** class, but it can be used to define an abstract set. This class contains a concrete set, which can be initialized by the **load()** method, or directly.

Constructor arguments:

- within - A set that defines the type of values that can be contained in this set
- default - Default set members, which may be overridden when setting up this set
- rule - A rule for setting up this set with existing model data. This has the functional form: f: pyomo.Model $->$ pyomo.Set
- restriction - Define a rule for restricting membership in a set. This has the functional form: f: data $->$ bool and returns true if the data belongs in the set

**3.2. Product Sets.** The **ProductSet()** class represents the cross product of other sets.

Constructor arguments:

- default - Default set members, which may be overridden when setting up this set
- rule - A rule for setting up this set with existing model data. This has the functional form: f: pyomo.Model $->$ pyomo.Set
- restriction - Define a rule for restricting membership in a set. This has the functional form: f: data $->$ bool and returns true if the data belongs in the set

In the following AMPL code, the **rate** parameter's index set is the cross product of two sets:

```
set PROD;
set STAGE;

param rate {PROD,STAGE};
```

In Pyomo, the cross product is created with the **ProductSet** class, and the result of this is used to index other Pyomo objects:

```
model.PROD = Set()
model.STAGE = Set()

model.setprod = ProductSet( (model.PROD, model.STAGE) )
model.rate = Param(index=model.setprod)
steel4mod.rate > 0
```

**3.3. Parameters.** The **Param()** class defines constant values in a model, and a parameter object may be defined over an index.

Constructor arguments :

- index - The index set that defines the distinct parameters. By default, this is None, indicating that there is a single parameter.
- domain - A set that defines the type of values that each parameter must be.
- validate - A rule for validating this parameter with respect to data that exists in the model
- default - A set that defines default values for this parameter
- rule - A rule for setting up this parameter with existing model data

**3.4. Variables.** The **Var()** class defines a numeric variable, which may be defined over an index.

Constructor arguments:

- index - The index set that defines the distinct variables. By default, this is None, indicating that there is a single variable.
- domain - A set that defines the type of values that each parameter must be.
- default - A set that defines default values for this variable
- bounds - A function that defines bound constraints for this variable

Simple bound constraints on variables can be specified with the **bounds** rule:

```
model.P    = Set()

model.x_lb = Param(index=model.P)
model.x_ub = Param(index=model.P)

def x_bounds(i, model):
    return (model.x_lb[i], model.x_ub[i])
model.x = Var(index=model.P, rule=x_bounds)
```

**3.5. Objectives.** The **Objective()** class defines an objective expression.

Constructor arguments:

- rule - A rule for constructing this objective with existing model data.
- sense - Used to define wether this objective should be minimized or maximized (minimization is the default).

**3.6. Constraints.** The **Constraint()** class defines an expression whose value is constrained in the model.

Constructor arguments:

- rule - A rule for constructing this constraint with existing model data.
- index - Defines a set of constraints over an index.

Note that the **rule** option generally needs to include a definition of the bounds on a constraint. A constraint must have either an upper or lower bound, and it may have both. For example:

```
model.P = Set()
model.Q = Set()

model.x = Var(index=model.Q)

def c_rule(i, model):
    ans = 0
    for q in model.Q:
      ans = ans + model.x[q]
    ans = ans > 0
    return ans < 1
model.c = Constraint(index=model.P, rule=c_rule)
```

The last two lines in the **c_rule** function define upper and lower bound values for the **c** constraint. Note that this is a non-standard use of the < and > operators; these operators return an expression rather than a boolean value.

**3.7. Models.** The **Model()** class defines a mixed-integer model that can be optimized by a user. This class takes no arguments, but it is a container for instances of the other Pyomo objects created by the user. For example, consider the statement:

```
model.x = Var()
```

This statement registers the variable $x$ in the model, and assigns it the name “x”.

**4. Conclusions.** Pyomo has many of the features of abstract modeling languages and optimization modeling libraries, but the following features of Pyomo are noteworthy:

- Pyomo supports the ability to define abstract problems from which problem instances can be generated. Further, Pyomo can generate multiple instances, which can be analyzed simultaneously in separate Python class objects.
- Pyomo is based on a powerful, commonly available open-source language. Thus, there are no licensing limitations with the use of Pyomo, and the set of Pyomo objects can be customized for an application in ways that are not possible with commercial AMLs and modeling libraries.
- Python has a clean syntax, so Pyomo modeling objects can be used in an intuitive manner.
- Pyomo models can leverage Python's programming language to define complex data structures and standard programming constructs like classes and functions. Further, Python can be naturally linked with external libraries for high-performance kernels.
- Pyomo can integrate optimization solvers in an extensible manner. Optimizers can be defined within Python itself, and external optimizers can be launched using file I/O to communicate with Python.[1]

Pyomo is probably most similar to the FLOPC++ modeling library. FLOPC++ is writen in C++, and it has many of the same objects as are used in Pyomo. While FLOPC++ enables models to be embedded in compiled application codes, Pyomo enables the rapid prototyping of models in a scripting language. Thus, these capabilities seem quite complementary.

The current implementation of Pyomo has been validated on a small set of simple models. In the future, more extensive validation of Pyomo is needed to ensure that it can express a wide range of complex problems. This includes the integration of hooks for other types of optimizers, like the nonlinear optimizers in Dakota. Further, the performance of Pyomo needs to be analyzed to ensure that it can effectively generate large-scale optimization models. Finally, this document needs to be extended to include examples that illustrate how Pyomo can leverage Python to develop complex models more naturally than AMLs like AMPL and GAMS.

This document describes the initial prototype of Pyomo. Once this code has stabilized, we plan to integrate Pyomo into the COIN-OR optimization software repository to encourage

[1]For example, this is similar to the manner in which AMPL launches optimizers.

its use within the academic and business communities.

**Acknowledgements.** Thanks to Jon Berry and Cindy Phillips for their critical feedback on the design of Pyomo.

## REFERENCES

[1] *AIMMS home page*. `http://www.aimms.com`.
[2] *AMPL home page*. `http://www.ampl.com/`.
[3] *CVXOPT home page*. `http://abel.ee.ucla.edu/cvxopt`.
[4] *FLOPC++ home page*. `https://projects.coin-or.org/FlopC++`.
[5] *GAMS home page*. `http://www.gams.com`.
[6] *OPL home page*. `http://www.ilog.com/products/oplstudio`.
[7] *Pulp: A python linear programming modeler*. `http://www.jeannot.org/~js/code/index.en.html`.
[8] R. Fourer, D. M. Gay, and B. W. Kernighan, *AMPL: A Modeling Language for Mathematical Programming, 2nd Ed.*, Brooks/Cole–Thomson Learning, Pacific Grove, CA, 2003.
[9] J. Kallrath, *Modeling Languages in Mathematical Optimization*, Kluwer Academic Publishers, 2004.

```
# Imports
from pyomo import *

# Setup the model
model = Model()

model.P = Set()

model.a = Param(index=model.P)
model.b = Param()
model.c = Param(index=model.P)
model.u = Param(index=model.P)

def X_bounds(j, model):
    return (0,model.u[j])
model.X = Var(index=model.P, bounds=X_bounds)

def Objective_rule(model):
    ans = 0
    for j in model.P:
      ans = ans + model.c[j] * model.X[j]
    return ans
model.profit = Objective(rule=Objective_rule)

def Time_rule(model):
    ans = 0
    for j in model.P:
      ans = ans + (1.0/model.a[j]) * model.X[j]
    return ans < model.b
model.Time = Constraint(rule=Time_rule)

print "ABSTRACT MODEL"
model.pprint()

# Create the model instance
instance = model.create("prod.dat")

print "MODEL INSTANCE"
instance.pprint()
```

## Transformation

In the section we include research in areas transforming the way science and engineering are conducted. Much effort in the past has been focused solely on the creation of computational models, also known as the forward model. This approach alone is no longer sufficient as the performance, reliability, and safety requirements for systems depend critically the results of these models. Techniques and procedures are needed to assess the credibility of computational simulations for engineering and scientific analyses if we are to use them for high-level decision-making. The papers in this section address issues and applications in validation & verification, uncertainty quantification, and related areas.

Vanderlei and Hopkins consider *a posteriori* error estimation for immersed interface solutions to quantify the error introduced in the problem discretization. Shunn and Knupp explore the effects of tabulated state-relationships on the computational performance of low-Mach number combustion codes using the method of manufactured solutions (MMS). In addition to verifying the order-of-accuracy of the code, the MMS helps highlight robustness issues in existing variable-density flow-solvers. Varela and Oldfield describe their efforts to validate analytical models describing the impact of checkpointing on the performance of large-scale applications on massive-scale supercomputers. Reale-Levis *et al.* describe a new algorithm for optimization under uncertainty that simultaneously minimizes the variance of a desired output response while maintaining the constraint that the mean is fixed at a certain level. They then demonstrate their algorithm using an automotive device design robustness example. McFarland *et al.* explored the use of Bayesian model calibration as a tool for calibrating a computational simulation with experimental observations. Their methodology simultaneously keeps track of the uncertainty in the process. Erten and Knupp describe a mesh-optimization algorithm for curved domains utilizing a target-matrix paradigm. Finally, Constantine and Knupp develop a stochastic method of manufactured solutions. This allows them to verify the convergence of statistics of the solutions to stochastic PDEs produced by uncertainty quantification algorithms that compute moments of solutions of PDEs with random inputs.

M.L. Parks
S.S. Collis

December 6, 2007

# ERROR ESTIMATION FOR IMMERSED INTERFACE SOLUTIONS

BENJAMIN A. VANDERLEI[†] AND MATTHEW M. HOPKINS[‡]

**Abstract.** We apply a modification of the Method of Nearby Problems in order to compute error estimates for solutions obtained using the Immersed Interface Method. The problem we examine is an elliptic PDE in which the coefficients have discontinuities across some internal boundary.

**1. Introduction.** In order to make numerical approximations more useful in application settings it is important to understand and, if possible, to quantify the error introduced by discretizing the problem. An important class of error estimators known as a posteriori estimates attempt to quantify the error in a particular solution given only the solution. This work uses a variation of the Method of Nearby Problems [1, 4, 5, 6, 7], which is related to the defect correction principle. We consider the technique applied to an elliptic problem in which the coefficients have discontinuities across some internal boundary, for which we use the Immersed Interface Method [2, 3] to obtain numerical solutions.

**2. Method of Nearby Problems.** The Method of Nearby Problems (MNP) is a post-processing technique used to compute an estimate of the discretization error associated with a numerical solution. The key idea of MNP is to construct a problem which is "nearby" the problem of interest, yet is one for which an exact solution is known. The question of how best to construct this perturbation, and the resulting definition of the "nearness" of the problem are currently under investigation.

We describe here an example of carrying out the MNP procedure in a discrete setting as opposed to approaches using solution reconstruction. Consider the following linear elliptic boundary value problem: $Lu = f$ and the finite difference discretization: $L_h u_h = f_h$ which one solves to obtain the numerical approximation $u_h$. The defect correction principle [8] is a very general idea that allows one to describe an iterative solution method to a problem based on a series of corrections. Alternatively, one may apply this principle to generate an estimate of the error associated with a particular solution rather than use the information as a correction. We have implemented the procedure as follows:

1. Begin with solving the original discrete problem $L_h u_h = f_h$.
2. Construct a second discretization of the operator $L$, say $\tilde{L}_h$.
3. Use $\tilde{L}_h$ to compute the defect (perturbation) $s_h = \tilde{L}_h u_h - f_h$
4. Use the original discretization to solve a perturbed problem $L_h v_h = f_h + s_h$.
5. Estimate the discretization error of $u_h$ with the quantity $u_h - v_h$.

One difference between this technique and the original MNP is that in this setting we no longer have access to an exact analytic solution of the nearby problem and thus cannot measure the error directly in solving it. While we do not have an exact solution, we do have the original approximation $u_h$ to work with. In fact, one may write $u_h - v_h$ in terms of the true error in order to show that it may be a good estimate.

We will use the fact that $L_h v_h = f_h + s_h = \tilde{L}_h u_h$ together with the following definitions of the local truncation errors $\tau_h$ and $\tilde{\tau}_h$ of $L_h$ and $\tilde{L}_h$ respectively:

$$L_h u_R = f_h + \tau_h \tag{2.1}$$

$$\tilde{L}_h u_R = f_h + \tilde{\tau}_h \tag{2.2}$$

where $u_R$ is the analytic solution of the original problem restricted to the grid.

[†]Tulane University, bvander@math.tulane.edu

[‡]Sandia National Laboratories, mmhopki@sandia.gov

$$\begin{aligned}
L_h v_h &= f_h + s_h \\
&= \tilde{L}_h u_h && \text{(definition of } s_h\text{)} \\
&= \tilde{L}_h L_h^{-1} f_h && (u_h \text{ solves original problem}) \\
&= \tilde{L}_h (u_R - L_h^{-1}\tau_h) && \text{(by 2.1)} \\
&= f_h + \tilde{\tau}_h - \tilde{L}_h L_h^{-1}\tau_h && \text{(by 2.2)} \\
&= L_h u_R - \tau_h + \tilde{\tau}_h - \tilde{L}_h L_h^{-1}\tau_h && \text{(by 2.1)}
\end{aligned}$$

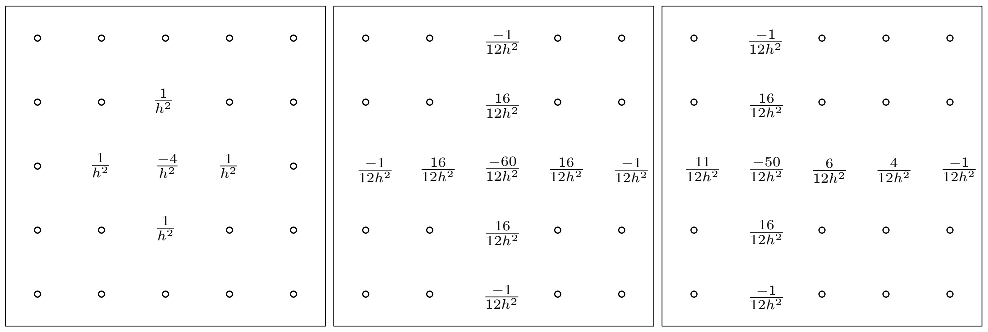
so that

$$v_h = u_R - L_h^{-1}\tau_h + L_h^{-1}\tilde{\tau}_h - L_h^{-1}\tilde{L}_h L_h^{-1}\tau_h$$

The estimate of the error can then be written:

$$\begin{aligned}
u_h - v_h &= u_h - (u_R - L_h^{-1}\tau_h + L_h^{-1}\tilde{\tau}_h - L_h^{-1}\tilde{L}_h L_h^{-1}\tau_h) \\
&= -(u_R - u_h) + L_h^{-1}\tau_h - L_h^{-1}\tilde{\tau}_h + L_h^{-1}\tilde{L}_h L_h^{-1}\tau_h \\
&= L_h^{-1}\tilde{L}_h L_h^{-1}\tau_h - L_h^{-1}\tilde{\tau}_h \\
&= L_h^{-1}\tilde{L}_h (u_R - u_h) - L_h^{-1}\tilde{\tau}_h
\end{aligned}$$

We see then that $u_h - v_h$ will be a good approximation of the true error provided that $L_h^{-1}\tilde{\tau}$ is small and $L_h^{-1}\tilde{L}_h$ is an approximation to the identity operator. A natural choice for $\tilde{L}_h$ is a higher order discretization of $L$.


$\frac{1}{h^2}$
$\frac{1}{h^2}$ $\frac{-4}{h^2}$ $\frac{1}{h^2}$
$\frac{1}{h^2}$
$\frac{-1}{12h^2}$
$\frac{16}{12h^2}$
$\frac{-1}{12h^2}$ $\frac{16}{12h^2}$ $\frac{-60}{12h^2}$ $\frac{16}{12h^2}$ $\frac{-1}{12h^2}$
$\frac{16}{12h^2}$
$\frac{-1}{12h^2}$
$\frac{-1}{12h^2}$
$\frac{16}{12h^2}$
$\frac{11}{12h^2}$ $\frac{-50}{12h^2}$ $\frac{6}{12h^2}$ $\frac{4}{12h^2}$ $\frac{-1}{12h^2}$
$\frac{16}{12h^2}$
$\frac{-1}{12h^2}$

(a) standard 5 point stencil (b) wide 9 point stencil (c) boundary stencil

FIG. 2.1. *Stencils used for* $\tilde{L}_h$

To give an example of the estimates that can be obtained we consider a Poisson problem on a square domain with Dirichlet boundary conditions. For the discrete operator $L_h$ we will use the standard 5-point finite difference scheme shown in Figure 2.1(a) applied on a regular $N$x$N$ grid. To construct $\tilde{L}_h$ we take the sum of one-dimensional high-order difference approximations of the second derivative. The stencil and corresponding coefficients for this fourth order scheme are shown in Figure 2.1(b). Note that for grid points adjacent to a boundary we make use of asymmetric stencils such as the one shown in Figure 2.1(c).

Figure 2.2 shows the results. We see that the discrepancy between the estimate and the true error converges to zero as $O(h^4)$.

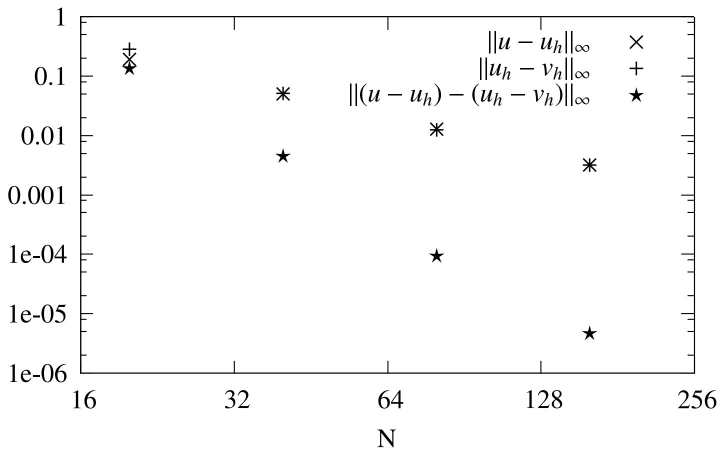


FIG. 2.2. *Error in estimates made for Poisson problem.*

**3. Interface Problem.** We next consider the following elliptic interface problem:

$$\nabla \cdot (\beta \nabla u) = f \tag{3.1}$$

on the domain $\Omega$ = [-1,1] x [-1,1] with Dirichlet boundary conditions. We will take the circle given by $x^2 + y^2 = 1/4$ as the interface between the subdomains and define $\Omega^-$ to be the set of points inside the circle and $\Omega^+$ to be the set of points outside the circle. Figure 2.3 shows this geometry. In each of the subdomains $\beta$ will be constant. In §3.1 we will describe how to obtain a numerical solution using the Immersed Interface Method [2]. In §3.2 we describe the construction of a higher order discretization that we will use as $\tilde{L}_h$.

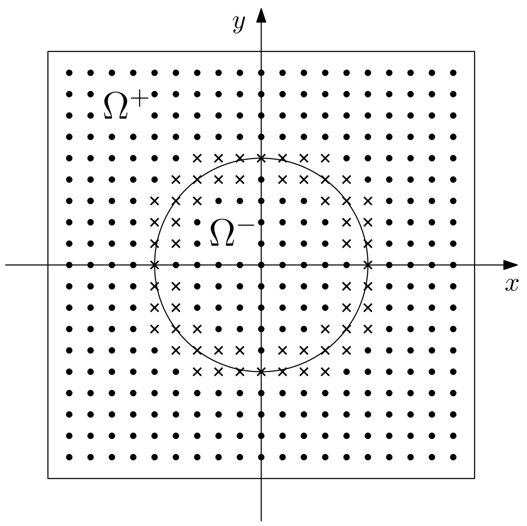


FIG. 2.3. *Geometry of interface problem with irregular points indicated.*

**3.1. Immersed Interface Method.** The methodology of the Immersed Interface Method (IIM) is to work on a regular Cartesian grid and use a standard discretization of the problem for points that are not near the interface. For our problem this means using the 5-point scheme (Figure 2.1(a)) at any point in the domain for which the stencil does not cross the interface. These points will be referred to as regular points. At the other grid points, termed irregular points, information about the problem is used to derive a special finite difference equation. The irregular points for the sample grid in Figure 2.3 are marked.

We will use a slight variation of the original IIM that preserves the maximum principle [3]. The aim is to derive a finite difference equation at an irregular point $(x_i, y_j)$ such as the following:

$$\sum_{k=1}^{9} c_k U_k = f_{i,j} \tag{3.2}$$

where $U_k$ are the unknown solution values at the grid points in the standard compact 9-point stencil shown in Figure 3.2(a), $c_k$ are the coefficients to be determined, and $f_{i,j} = f(x_i, y_j)$. The derivation of the scheme is as follows:

1. Express the solution at all points in the stencil as a Taylor series centered about some point on the interface.
2. Determine interface relations for the solution $u$ and its derivatives.
3. Determine the local truncation error using the interface relations to express everything in terms of quantities from one side of the interface.
4. Determine the system of equations for the coefficients that must be satisfied to eliminate the leading order terms.
5. Solve a least squares optimization problem to find the coefficients $c_k$.

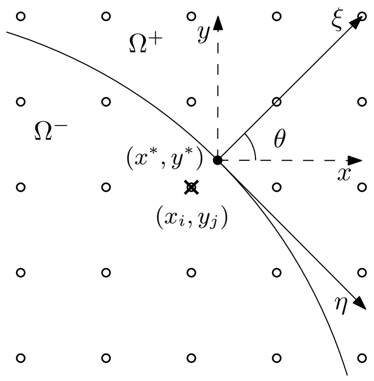
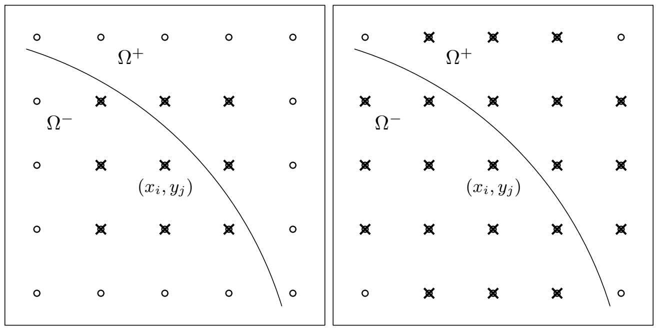


FIG. 3.1. *Local coordinate system.*

We first require a point on the interface about which to write the Taylor series. We will assume there exists a unique point closest to $(x_i, y_j)$ and call this point $(x^*, y^*)$. For the purposes of writing the interface relations, it is convenient to work in a local coordinate system defined by the normal and tangential directions of the interface at the point $(x^*, y^*)$. This coordinate transformation is given by:

$$\xi = (x - x^*)\cos\theta + (y - y^*)\sin\theta$$
$$\eta = -(x - x^*)\sin\theta + (y - y^*)\cos\theta$$

where $\theta$ is the angle between the $x$-axis and the normal direction $\xi$ as shown in Figure 3.1.



(a) 9 point stencil (b) 21 point stencil

FIG. 3.2. *Stencils used for $L_h$ and $\tilde{L_h}$ at irregular point $(x_i, y_j)$.*

There are two interface conditions we wish to express in local coordinates, a continuity condition and a jump specified in the normal derivative:

$$u^+ = u^- \tag{3.3}$$

$$(\beta u_n)^+ = (\beta u_n)^- \tag{3.4}$$

Note that these conditions are derived directly from (3.1).

We will express the interface locally as $\xi = \chi(\eta)$. Note that at the point $(x^*, y^*)$ we have $\eta = 0$ and $\chi(0) = 0$. Also if the interface is smooth at $(x^*, y^*)$, $\chi'(0) = 0$. The continuity condition is now $u^+(\chi(\eta), \eta) = u^-(\chi(\eta), \eta)$. One may now differentiate this condition with respect to $\eta$ and, after evaluating at $(x^*, y^*)$, obtain:

$$u^+_\eta = u^-_\eta$$

Differentiating once more with respect to $\eta$ and evaluating at $(x^*, y^*)$ gives:

$$u^+_{\eta\eta} + \chi'' u^+_\xi = u^-_{\eta\eta} + \chi'' u^-_\xi$$

Next we rewrite the condition on the normal derivative in local coordinates. The normal derivative is given by:

$$u^\pm_n = \frac{1}{\sqrt{1 + \chi'^2}}(u^\pm_\xi - u^\pm_\eta \chi')$$

Substituting this representation into (3.4) gives us the relation:

$$\beta^+ u^+_\xi = \beta^- u^-_\xi$$

As with the tangential condition, one may differentiate this relation with respect to $\eta$ and evaluate at $(x^*, y^*)$, which gives:

$$\beta^+(u^+_{\xi\eta} - u^+_\eta \chi'') = \beta^-(u^-_{\xi\eta} - u^-_\eta \chi'')$$

In order to get the final relation needed we use the fact that (3.1) is unchanged by the change of coordinates, and the assumed continuity of $f$, which means that $f^+ = f^-$. The final relation then is:

$$\beta^+(u^+_{\xi\xi} + u^+_{\eta\eta}) = \beta^-(u^-_{\xi\xi} + u^-_{\eta\eta})$$

In preparation of expressing the truncation error in terms of quantities from only one side of the interface we define $\rho = \beta^-/\beta^+$ and rewrite the above conditions to give all plus quantities in terms of minus quantities:

$$\begin{aligned}
u^+ &= u^- \\
u^+_\eta &= u^-_\eta \\
u^+_\xi &= \rho u^-_\xi \\
u^+_{\xi\eta} &= \rho u^-_{\xi\eta} + (1-\rho)u^-_\eta \chi'' \\
u^+_{\eta\eta} &= u^-_{\eta\eta} + (1-\rho)u^-_\xi \chi'' \\
u^+_{\xi\xi} &= \rho u^-_{\xi\xi} + (\rho - 1)u^-_{\eta\eta} + (\rho - 1)u^-_\xi \chi''
\end{aligned} \tag{3.5}$$

With all interface relationships in place, we are ready to write down the truncation error of the special scheme at irregular point $(x_i, y_j)$.

$$T_{i,j} = \sum_{k=1}^{9} [c_k u(\xi_k, \eta_k)] - f_{i,j}$$

We next expand $u$ at the grid points in a Taylor series about $(x^*, y^*)$:

$$u(\xi_k, \eta_k) = u^\pm + \xi_k u^\pm_\xi + \eta_k u^\pm_\eta + \frac{1}{2}\xi_k^2 u^\pm_{\xi\xi} + \frac{1}{2}\eta_k^2 u^\pm_{\eta\eta} + \xi_k \eta_k u^\pm_{\xi\eta} + O(h^3)$$

Again we also note that $f_{i,j} = f(x^*, y^*) + O(h)$. We now define index sets $K^\pm$ by:

$$K^\pm = \{k : (\xi_k, \eta_k) \in \Omega^\pm\}$$

and then write $T_{i,j}$ as:

$$\begin{aligned}
T_{i,j} = {} & a_1 u^- + a_2 u^+ + a_3 u^-_\xi + a_4 u^+_\xi + a_5 u^-_\eta + a_6 u^+_\eta + a_7 u^-_{\xi\xi} + a_8 u^+_{\xi\xi} + \\
& a_9 u^-_{\eta\eta} + a_{10} u^+_{\eta\eta} + a_{11} u^-_{\xi\eta} + a_{12} u^+_{\xi\eta} - f(x^*, y^*) + O(h)
\end{aligned}$$

where

$$\begin{aligned}
a_1 &= \sum_{k\in K^-} c_k & a_2 &= \sum_{k\in K^+} c_k \\
a_3 &= \sum_{k\in K^-} c_k \xi_k & a_4 &= \sum_{k\in K^+} c_k \xi_k \\
a_5 &= \sum_{k\in K^-} c_k \eta_k & a_6 &= \sum_{k\in K^+} c_k \eta_k \\
a_7 &= \frac{1}{2}\sum_{k\in K^-} c_k \xi_k^2 & a_8 &= \frac{1}{2}\sum_{k\in K^+} c_k \xi_k^2 \\
a_9 &= \frac{1}{2}\sum_{k\in K^-} c_k \eta_k^2 & a_{10} &= \frac{1}{2}\sum_{k\in K^+} c_k \eta_k^2 \\
a_{11} &= \sum_{k\in K^-} c_k \xi_k \eta_k & a_{12} &= \sum_{k\in K^+} c_k \xi_k \eta_k
\end{aligned}$$

We now use the interface relations (3.5) to replace all plus quantities in $T_{i,j}$ with minus quantities and replace $f(x^*, y^*)$ with $\beta^-(u^-_{\xi\xi} + u^-_{\eta\eta})$. Collecting terms gives:

$$\begin{aligned} T_{i,j} &= (a_1 + a_2)u^- + (a_3 + \rho a_4 + (\rho - 1)a_8\chi'' + (1 - \rho)a_{10}\chi'')u^-_\xi \\ &\quad + (a_5 + a_6 + (1 - \rho)a_{12}\chi'')u^-_\eta + (a_7 + \rho a_8 - \beta^-)u^-_{\xi\xi} \\ &\quad + (a_9 + a_{10} + (\rho - 1)a_8 - \beta^-)u^-_{\eta\eta} + (a_{11} + \rho a_{12})u^-_{\xi\eta} + O(h) \end{aligned}$$

It is enough to achieve first order accuracy at the irregular points in order to recover second order global accuracy in the solution [3]. In order to force $T_{i,j}$ to be first order the coefficients must satisfy the following six equations:

$$\begin{array}{cc} a_1 + a_2 = 0 & a_3 + \rho a_4 + a_8(\rho - 1)\chi'' + a_{10}(1 - \rho)\chi'' = 0 \\ a_5 + a_6 + a_{12}(1 - \rho)\chi'' = 0 & a_7 + \rho a_8 = \beta^- \\ a_9 + a_{10} + a_8(\rho - 1) = \beta^- & a_{11} + \rho a_{12} = 0 \end{array}$$

The final step is to solve an optimization problem for the 9 unknown coefficients subject to the 6 constraints just derived. We minimize the difference between the $c_k$'s and the standard coefficients (Figure 2.1(a)) in the least squares sense.

**3.2. Higher Order Discretization.** We next describe the construction of $\tilde{L}_h$, the higher order version of the discretization just described. As in the Poisson problem test case we will use the wide 9-point stencil (Figure 2.1(b)) to achieve a fourth order truncation error at regular points. We follow the same methodology as before to derive the high order scheme for the irregular points but include more terms in the series expansions.

The usual idea to achieve a higher order truncation error for the irregular points is to force the next order terms in the series expansion to be zero. There are 4 third order terms, so we expect to have 10 equations total. Experimentation has shown that the stencil of 21 points shown in Figure 3.2(b) yields the most consistent solution to the optimization problem. Using fewer points tends to degrade the results when the discontinuities in $\beta$ become large.

The first component necessary is a set of interface relations for the third order derivatives so that all plus quantities may be written in terms of minus quantities. To this end we differentiate the continuity and normal derivative conditions once more with respect to $\eta$ (along the interface) and evaluate at the point $(x^*, y^*)$ This yields the following two conditions for $u_{\xi\eta\eta}$ and $u_{\eta\eta\eta}$ in terms of the lower order derivatives:

$$\begin{gathered} \beta^+(u^+_{\xi\eta\eta} - 2u^+_{\eta\eta}\chi'' + u^+_{\xi\xi}\chi'') = \beta^-(u^-_{\xi\eta\eta} - 2u^-_{\eta\eta}\chi'' + u^-_{\xi\xi}\chi'') \\ u^+_{\eta\eta\eta} + 3u^+_{\xi\eta} = u^-_{\eta\eta\eta} + 3u^-_{\xi\eta} \end{gathered}$$

(Note that we neglected terms involving higher order derivatives of $\chi$ since $\chi''' = 0$ for this particular geometry.)

To get the other two condition needed we differentiate (3.1) with respect to $\xi$ and $\eta$ and again assume sufficient regularity of $f$ to get the following conditions.

$$\begin{gathered} \beta^+(u^+_{\xi\xi\xi} + u^+_{\xi\eta\eta}) = \beta^-(u^-_{\xi\xi\xi} + u^-_{\xi\eta\eta}) \\ \beta^+(u^+_{\xi\xi\eta} + u^+_{\eta\eta\eta}) = \beta^-(u^-_{\xi\xi\eta} + u^-_{\eta\eta\eta}) \end{gathered}$$

These relations allows us to write $u_{\xi\xi\xi}$ and $u_{\xi\xi\eta}$ in terms of the other third derivatives.

The truncation error is again:

$$
\begin{aligned}
T_{i,j} &= \sum_{k=1}^{21}\left[c_k u(\xi_k,\eta_k)\right] - f_{i,j} \\
&= a_1 u^- + a_2 u^+ a_3 u_\xi^- a_4 + u_\xi^+ + a_5 u_\eta^- + a_6 u_\eta^+ + a_7 u_{\xi\xi}^- + a_8 u_{\xi\xi}^+ \\
&+ a_9 u_{\eta\eta}^- + a_{10} u_{\eta\eta}^+ + a_{11} u_{\xi\eta}^- + a_{12} u_{\xi\eta}^+ a_{13} u_{\xi\xi\xi}^- + a_{14} u_{\xi\xi\xi}^+ \\
&+ a_{15} u_{\eta\eta\eta}^- + a_{16} u_{\eta\eta\eta}^+ + a_{17} u_{\xi\xi\eta}^- + a_{18} u_{\xi\xi\eta}^+ + a_{19} u_{\xi\eta\eta}^- + a_{20} u_{\xi\eta\eta}^+ \\
&+ -[f(x^*,y^*) + \xi_1 f_\xi(x^*,y^*) + \eta_1 f_\eta(x^*,y^*)] + O(h^2)
\end{aligned}
$$

where

$$
\begin{aligned}
a_{13} &= \frac{1}{6}\sum_{k\in K^-} c_k \xi_k^3 & a_{14} &= \frac{1}{6}\sum_{k\in K^+} c_k \xi_k^3 \\
a_{15} &= \frac{1}{6}\sum_{k\in K^-} c_k \eta_k^3 & a_{16} &= \frac{1}{6}\sum_{k\in K^+} c_k \eta_k^3 \\
a_{17} &= \frac{1}{2}\sum_{k\in K^-} c_k \xi_k^2 \eta_k & a_{18} &= \frac{1}{2}\sum_{k\in K^+} c_k \xi_k^2 \eta \\
a_{19} &= \frac{1}{2}\sum_{k\in K^-} c_k \xi_k \eta_k^2 & a_{20} &= \frac{1}{2}\sum_{k\in K^+} c_k \xi_k \eta^2
\end{aligned}
$$

Again we replace all plus quantities with minus quantities and replace $f_\xi$ and $f_\eta$ with $\beta^-(u_{\xi\xi\xi}^- + u_{\xi\eta\eta}^-)$ and $\beta^-(u_{\xi\xi\eta}^- + u_{\eta\eta\eta}^-)$ respectively. Gathering the terms and rewriting gives:

$$
\begin{aligned}
T_{i,j} &= (a_1 + a_2)u^- \\
&+ (a_3 + \rho a_4 + (\rho-1)a_8\chi'' + (1-\rho)a_{10}\chi'' + 3a_{14}(\rho-1)\chi''^2 + 3a_{20}(1-\rho)\chi''^2)u_\xi^- \\
&+ (a_5 + a_6 + (1-\rho)a_{12}\chi'' + 3a_{16}(\rho-1)\chi''^2 + 3a_{18}(1-\rho)\chi''^2)u_\eta^- \\
&+ (a_7 + \rho a_8 - \beta^-)u_{\xi\xi}^- \\
&+ (a_9 + a_{10} + (\rho-1)a_8 + 3a_{14}(\rho-1)\chi'' + 3a_{20}(1-\rho)\chi'' - \beta^-)u_{\eta\eta}^- \\
&+ (a_{11} + \rho a_{12} + 3a_{16}(1-\rho)\chi'' + 3a_{18}(\rho-1)\chi'')u_{\xi\eta}^- \\
&+ (a_{13} + \rho a_{14} - \beta^-\xi_1)u_{\xi\xi\xi}^- + (a_{15} + a_{16} + a_{18}(\rho-1) - \beta^-\eta_1)u_{\eta\eta\eta}^- \\
&+ (a_{17} + \rho a_{18} - \beta^-\eta_1)u_{\xi\xi\eta}^- + (a_{19} + \rho a_{20} - \beta^-\xi_1)u_{\xi\eta\eta}^- + O(h^2)
\end{aligned}
$$

Now we may write the following system of equations which should be satisfied by the coefficients $c_k$:

$$\begin{aligned}
a_1 + a_2 &= 0\\
a_3 + \rho a_4 + a_8(\rho-1)\chi'' + a_{10}(1-\rho)\chi'' + 3a_{14}(\rho-1)\chi''^2 + 3a_{20}(1-\rho)\chi''^2 &= 0\\
a_5 + a_6 + a_{12}(1-\rho)\chi'' + 3a_{16}(\rho-1)\chi''^2 + 3a_{18}(1-\rho)\chi''^2 &= 0\\
a_7 + \rho a_8 &= \beta^-\\
a_9 + a_{10} + a_8(\rho-1) + 3a_{14}(\rho-1)\chi'' + 3a_{20}(1-\rho)\chi'' &= \beta^-\\
a_{11} + \rho a_{12} + 3a_{16}(1-\rho)\chi'' + 3a_{18}(\rho-1)\chi'' &= 0\\
a_{13} + \rho a_{14} &= \beta^-\xi_1\\
a_{15} + a_{16} + a_{18}(\rho-1) &= \beta^-\eta_1\\
a_{17} + \rho a_{18} &= \beta^-\eta_1\\
a_{19} + \rho a_{20} &= \beta^-\xi_1
\end{aligned}$$

As in §3.1 we now minimize the difference between $c_k$'s and the coefficients used at regular points (Figure 2.1(b)), subject to the derived constraints. Note that $(\xi_1, \eta_1)$ are the local coordinates of the point $(x_i, y_j)$.

**4. Error Estimates for the Interface Problem.** We take as our test problem (3.1) with $f = -\pi^2 \cos(\pi r) - \pi \sin(\pi r)/r$, $\beta^- = 1$, and $\beta^+ = B$, where $r = \sqrt{x^2+y^2}$. The analytic solution is then $u^- = \cos(\pi r)$ and $u^+ = \cos(\pi r)/B$. Figures 4.1 and 4.2 show the results for solutions computed on uniform $N$x$N$ grids with $B = 10$ and $B = 100$ respectively. We see that the estimates are not only reliable for each of the grid sizes, but the difference between the error and the estimate is converging to zero. Note that for this problem the error in the estimate (discrepancy between the error and the estimate) converges to zero at the same rate as the true error. In the case of the continuous Poisson problem this error in the estimate converged to zero faster than the error. This is not unexpected if we recall that part of the error in the estimate is due to the size of $\tilde{\tau}_h$, the local truncation error of the discretization used to compute the perturbation. For the Poisson problem $\|\tau_h\|_\infty = O(h^2)$ and $\|\tilde{\tau}_h\|_\infty = O(h^4)$ while for the interface problem $\|\tau_h\|_\infty = O(h)$ and $\|\tilde{\tau}_h\|_\infty = O(h^2)$

We also compute the effectivity index of the estimates defined as:

$$\kappa = \frac{\|u_h - v_h\|}{\|u_R - u_h\|}$$

These results are shown in Table 4.1. The discrete version of the $L^2$ norm that we compute here is the following:

$$\|u_h\|_{L^2_h} = \left(\sum_{i=1}^{N}\sum_{j=1}^{N} u_h(x_i, y_j)\Delta x \Delta y\right)^{1/2}$$

We see that estimates of both the max norm and the $L^2_h$ norm are reasonable for all grid sizes.

The cost in producing these estimates is primarily a second solve of the associated linear system at the same grid size. One must consider however that some of the information from the solution of the original problem may be reused. For example if an iterative solver is being used, the original solution $u_h$ can be used as the initial guess. If a direct solve involving a factorization of the matrix is used, that factorization can be reused for the second solve. Table 4.2 shows the runtimes for the case $B = 10$. We observe that as the size of the problem

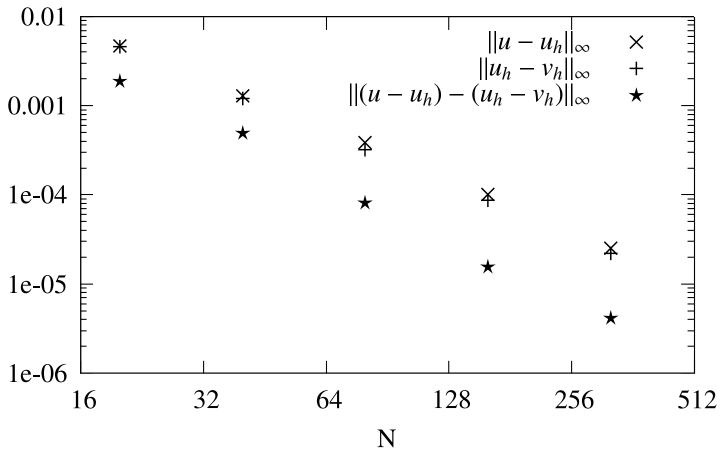


FIG. 4.1. *Error in estimates made for interface problem with $B = 10$.*

TABLE 4.1
*Effectivity indexes*

| | $B = 10$ | | $B = 100$ | |
|---|---|---|---|---|
| $N$ | $\|\| \cdot \|\|_\infty$ | $\|\| \cdot \|\|_{L_h^2}$ | $\|\| \cdot \|\|_\infty$ | $\|\| \cdot \|\|_{L_h^2}$ |
| 20 | 0.996 | 1.291 | 0.935 | 0.958 |
| 40 | 0.939 | 1.337 | 0.935 | 0.934 |
| 80 | 0.842 | 1.071 | 0.958 | 0.943 |
| 160 | 0.866 | 1.026 | 0.953 | 0.938 |
| 320 | 0.868 | 1.025 | 0.943 | 0.927 |

increases, the time taken to compute the error estimate becomes small relative to the time to compute the original solution. This is due to the fact that while the problems are small, the majority of the work goes into constructing the linear system (solving the optimization problems at irregular points.) As the problem size grows this cost becomes negligible relative to the cost of solving the system. Table 4.3 shows us just the time taken to solve the underlying linear system. As expected we see in all cases that it takes less time to solve the nearby problem than the original due to the improved initial guess.

TABLE 4.2
*Runtimes (in seconds) for the case $B = 10$.*

| $N$ | Compute Solution | Compute Estimate | $\frac{\text{Estimate time}}{\text{Solution time}}$ | $\frac{\text{Estimate time}}{\text{Total time}}$ |
|---|---|---|---|---|
| 20 | 1.07 | 4.60 | 4.34 | 0.813 |
| 40 | 2.67 | 8.95 | 3.35 | 0.770 |
| 80 | 14.0 | 22.5 | 1.61 | 0.616 |
| 160 | 528 | 209 | 0.395 | 0.284 |

**5. Conclusions and Future Directions.** We have computed reliable estimates of the numerical error in finite difference solutions of an elliptic problem with discontinuous coef-

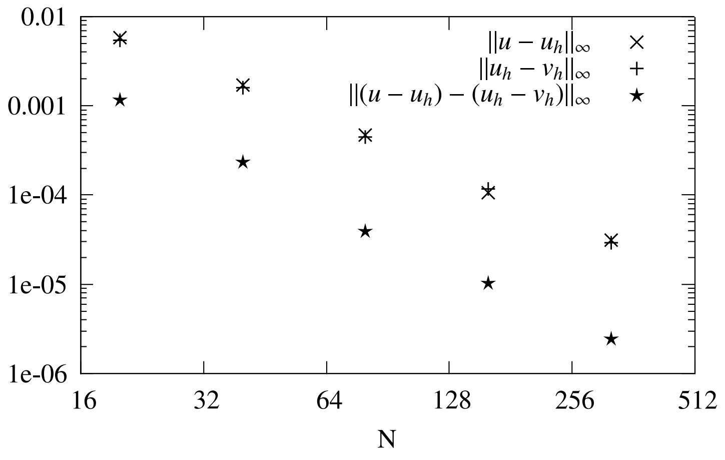


FIG. 4.2. *Error in estimates made for Interface problem with $B = 100$.*

TABLE 4.3
*GMRES solvetimes (in seconds) for the case $B = 10$.*

| $N$ | Solve Problem | Solve Nearby Problem |
|---|---|---|
| 20 | 0.115 | 0.098 |
| 40 | 0.808 | 0.428 |
| 80 | 10.1 | 5.3 |
| 160 | 517 | 169 |

ficients. The numerical solutions were obtained using the Immersed Interface Method and the estimates were computed using a procedure based on the Method of Nearby Problems. Future work will see this method extended to more general problems solved by IIM, such as the addition of singular source terms. One dimensional results indicate that a modification of our technique will be successful for such problems. We also wish to investigate the use of a compact high order scheme as the operator $\tilde{L}$. Such fourth order schemes have been derived and could potentially improve on the current estimates.

## REFERENCES

[1] M. Hopkins and C. Roy, *Introducing the method of nearby problems*, ECCOMAS, (2004).
[2] R. LeVeque and Z. Li, *The immersed interface method for elliptic equations with discontinuous coefficients and singular sources.*, SIAM J. Numerical Analysis, 31 (1994), pp. 1019–1044.
[3] Z. Li and K. Ito, *Maximum principle preserving schemes for interface problems with discontinuous coefficients*, SIAM J. Scientifice Computation, 23 (2001), pp. 339–361.
[4] A. Raju, *Discretization error estimation using the method of nearby problmes: One-dimensional cases*, master's thesis, Aerospace Engineering Dept. Auburn University, 2005.
[5] A. Raju, C. Roy, and M. Hopkins, *On the generation of exact solution using the method fo nearby problems*, AIAA, (2005).
[6] C. Roy and M. Hopkins, *Discretization error estimates using exact solutions to nearby problems*, AIAA, 0629 (2003).
[7] C. Roy, A. Raju, and M. Hopkins, *Estimation of discretization errors using the method of nearby problems*, AIAA, 45 (2007), pp. 1232–1243.
[8] H. Stetter, *The defect correction principle*, Numerishce Mathematik, 42 (1978), pp. 42–142.

# VERIFICATION OF LOW-MACH NUMBER COMBUSTION CODES USING THE METHOD OF MANUFACTURED SOLUTIONS

LEE SHUNN* AND PATRICK M. KNUPP†

**Abstract.** Many computational combustion models rely on tabulated constitutive relations to close the system of equations. As these reactive state-equations are typically multi-dimensional and highly non-linear, their implications on the convergence and accuracy of simulation codes are not well understood. In this report, the effects of tabulated state-relationships on the computational performance of low-Mach number combustion codes are explored using the method of manufactured solutions (MMS). The manufactured solutions are implemented in CDP, the multi-physics hydrodynamics code developed at Stanford University. Linear interpolation of the equation-of-state degrades convergence and introduces spurious fluctuations in the flow variables. In addition to verifying the order-of-accuracy of the code, the MMS problems help highlight robustness issues in existing variable-density flow-solvers.

## 1. Introduction.

The term *verification*, when applied to a computer code, describes the process of demonstrating that the code correctly solves its governing mathematical equations. A code that has been properly verified, therefore, is in likelihood free of programming errors which affect the theoretical order-of-accuracy of the numerical algorithm. As such, code verification is a integral step in building confidence in the predictive capabilities of simulation software.

Over the past several decades, the complexity of computational algorithms in simulation codes has grown in response to demands for high-fidelity simulations in science and engineering. State-of-the-art simulation codes often involve complex exchanges of information amongst various physics modules, each of which may solve different equations using different algorithms on different grid topologies. As simulation codes become more sophisticated, thorough verification becomes increasingly challenging and time consuming, yet also more essential.

The method of manufactured solutions (MMS) is a general procedure that can be used to construct analytic solutions to the differential equations that form the basis of a simulation code [21, 20, 22, 11, 13, 23]. The resulting solutions, while not necessarily physically relevant, can be used as benchmark solutions for verification tests. The accuracy of the code is gauged by running the test problems on systematically refined grids and comparing the output with the analytical manufactured solution. The behavior of the error can then be examined against the theoretical order-of-accuracy inherent in the code's numerical discretizations. Thus, a verification test using MMS provides an unambiguous result as to whether or not the algorithm is implemented correctly. MMS has been successfully applied in a variety of applications including fluid dynamics [24, 1], heat transfer [2, 3], fluid-structure interaction [27], even turbulence modeling [4].

In this work, attention is focused on hydrodynamics codes amenable to low-Mach number combustion where acoustic effects are unimportant. In this framework, a variable-density formulation of the Navier-Stokes equations is often used due to its computational efficiency relative to fully compressible formulations. In the variable-density equations, the pressure and density are formally decoupled by defining the density through an equation-of-state (EOS) expressed in terms of transported scalars. The EOS may be given by an analytic expression, or as is common for complex reactive systems, it may be precomputed and tabulated as a function of the scalars.

Tabulated state-equations are heavily used in many popular combustion models. Examples include laminar flamelet models [14, 15], conditional moment closure (CMC) meth-

*Stanford University, Mechanical Engineering, shunn@stanford.edu
†Sandia National Laboratories, pknupp@sandia.gov

ods [10], and some transported PDF methods [19]. These combustion models are used in a variety of combustion codes, targeting applications that include design and optimization of engines and power systems, prediction of pollutant formation in combustion devices, and modeling and prediction of fires. While validation studies of combustion codes are routinely performed, the application of systematic verification studies is less common. In particular, the ramifications of tabulated state-relationships on the convergence and accuracy of combustion codes has not been widely investigated. As the EOS in typical combustion systems is multi-dimensional and highly non-linear, its implications on code performance are not straightforward.

The objective of this work is to use MMS to explore the effects of tabulated constitutive relationships on the computational performance of low-Mach number combustion codes. Various example problems are developed and applied, progressing from simple one-dimensional configurations to examples involving higher dimensionality and solution-complexity. The new MMS problems can be used to complement existing verification tests that are suitable for the combustion codes described above. In addition to strict code verification, the MMS problems serve to highlight certain robustness issues and weaknesses in existing algorithms for variable-density flows.

**2. Governing equations.** The simulations in this report are performed using the unstructured large-eddy simulation (LES) code CDP [1]. The LES methodology recognizes that in many practical simulations, the governing equations admit solutions that cannot be resolved on affordably-sized grids. A filter (denoted by an overbar) is therefore introduced to separate the flow into resolved and unresolved scales. The large scales of motion are directly simulated while the smaller, dissipative scales are modeled. All field variables are decomposed into resolved and unresolved (subgrid or subfilter-scale) components using either a Reynolds decomposition

$$\rho = \bar{\rho} + \rho' \tag{2.1}$$

or a Favre (density-weighted) decomposition:

$$u_i = \tilde{u}_i + u_i'', \qquad \tilde{u}_i = \overline{\rho u_i}/\bar{\rho}. \tag{2.2}$$

Filtering the continuum Navier-Stokes equations yields equations for the resolved-scale variables: density $\bar{\rho}$, pressure $\bar{p}$, velocity $\tilde{u}_i$, and transported scalars $\tilde{\phi}_k$.

The combustion systems of interest in this work are characterized by relatively low Mach numbers ($Ma < 0.3$), hence, the assumption of negligible compressibility and acoustic effects is generally valid. The density is allowed to vary with the local temperature and species concentration (transported scalars), but is not a function of the local pressure.

Under these simplifications, variable-density reacting flows are described by the following conservation equations for mass, momentum, and species, combined with a suitable EOS:

$$\frac{\partial \bar{\rho}}{\partial t} + \frac{\partial \bar{\rho}\tilde{u}_j}{\partial x_j} = \bar{\tilde{S}}_{\bar{\rho}} \tag{2.3}$$

$$\frac{\partial \bar{\rho}\tilde{u}_i}{\partial t} + \frac{\partial \bar{\rho}\tilde{u}_j\tilde{u}_i}{\partial x_j} = -\frac{\partial \bar{p}}{\partial x_i} + \frac{\partial}{\partial x_j}\left(2\bar{\mu}\tilde{S}_{ij} - q_{ij}\right) + \bar{\tilde{S}}_{\tilde{u}_i} \tag{2.4}$$

[1]CDP is named after Dr. Charles David Pierce (1969-2002)

$$\frac{\partial \bar{\rho}\tilde{\phi}_k}{\partial t} + \frac{\partial \bar{\rho}\tilde{u}_j\tilde{\phi}_k}{\partial x_j} = \frac{\partial}{\partial x_j}\left(\bar{\rho}\tilde{\alpha}_k\frac{\partial \tilde{\phi}_k}{\partial x_j} - q_{\tilde{\phi}_k j}\right) + \bar{\tilde{S}}_{\tilde{\phi}_k} \tag{2.5}$$

$$\bar{\rho} = f(\tilde{\phi}_1, \tilde{\phi}_2, ...). \tag{2.6}$$

The resolved-scale stress is given by

$$\tilde{S}_{ij} = \frac{1}{2}\left(\frac{\partial \tilde{u}_i}{\partial x_j} + \frac{\partial \tilde{u}_j}{\partial x_i}\right) - \frac{1}{3}\delta_{ij}\frac{\partial \tilde{u}_k}{\partial x_k}, \tag{2.7}$$

and the subgrid (or subfilter) stress $q_{ij}$ and scalar fluxes $q_{\tilde{\phi}_k j}$ are modeled using the eddy-viscosity approach of Smagorinsky [26]:

$$q_{ij} = -2\mu_t\tilde{S}_{ij} \qquad \text{where} \qquad \mu_t = \bar{\rho}\, C\, \Delta^2|\tilde{S}|. \tag{2.8}$$

The unknown coefficient $C$ in (2.8) is closed using the dynamic procedure [6, 12, 7, 16]. In all simulations reported here, the codes are run in so-called "DNS" mode, wherein all subgrid models were disabled.

**3. Numerical method.** The computer code CDP uses a collocated, unstructured version of the algorithm of Pierce and Moin [17, 18]. This algorithm employs a temporally-staggered variable arrangement in which velocity components are staggered in time with respect to density and other scalar variables. The equations are spatially discretized using low-dissipation, node-based finite-volume operators developed by Ham *et al.* [8]. The variables are implicitly advanced in time using a fractional-step method, and an iterative approach is used at each time level to repair linearization errors and enhance stability. The major features of the iteration process at each time step are listed below. Here the superscript $m$ is used to denote the inner-iteration number.

1. The scalar equation(s) (2.5) are advanced in time. This yields $(\rho\phi)^{m+1}$, from which a provisional estimate for $\phi$ is obtained by $\widehat{\phi} = (\rho\phi)^{m+1}/\rho^m$.
2. The momentum equations (2.4) are advanced to obtain provisional velocities: $\widehat{u_i}$.
3. The provisional scalar values are used to evaluate the density from the EOS: $\rho^{m+1} = f(\widehat{\phi})$.
4. The updated density is used to correct the scalar(s) to ensure primary conservation: $\phi^{m+1} = (\rho\phi)^{m+1}/\rho^{m+1}$.
5. A constant-coefficient Poisson equation is solved for pressure, and the result is used to correct the velocity field to discretely conserve mass.
6. The process is repeated from step 1 and continued until convergence.

The linearized scalar and momentum equations (steps 1-2) are solved using a Jacobi method, and the Poisson solve (step 5) is accomplished using the HYPRE algebraic multigrid solver [5, 9]. Linear stability analysis indicates that the iterative approach outlined above is second-order accurate when at least two inner-iterations are employed [17]. Additional iterations may improve the stability of the scheme, but do not increase the order of accuracy. Formal verification of the second-order behavior of the algorithm requires convergence of the system at each time step.

**4. Example problems.** It has been noted that because code verification is a purely mathematical exercise, manufactured solutions need not be "realistic" [22]. While this is inarguably true, it also does not acknowledge the utility of well-crafted manufactured solutions in identifying the vulnerabilities and strengths of a computational algorithm. For instance, a manufactured solution that is *suggestive* of some elementary physics, provides not only a statement about the code's order-of-accuracy, but also gives a preview of how the code might perform in more complex problems where the mimicked physics are pervasive.

In this spirit, we introduce example MMS problems which attempt to illustrate "canonical" phenomena in variable-density flows. The examples are constructed such that they identically obey a subset of the governing physics without source terms (i.e. conservation of mass), and apply manufactured sources to satisfy the remaining conservation laws. Mass conservation is afforded preferential treatment in these examples due to its central role in the solution algorithm presented in Section 3. The resulting manufactured solutions attempt to balance simplicity with realism in an effort to understand how the code performs in "representative" scenarios.

The verification problems in this report are based on the EOS for isothermal binary mixing between miscible fluids:

$$\rho(\phi) = \left(\frac{\phi}{\rho_1} + \frac{1-\phi}{\rho_0}\right)^{-1}. \tag{4.1}$$

Although this EOS is simple, it is deceptively non-trivial. Large density ratios result in extremely non-linear behavior that can challenge variable-density solvers as much as the reactive state-equations associated with combustion chemistry. The scalar variable $\phi$ in (4.1) is known as the mixture fraction and assumes values ranging from 0 to 1. A similar mixture fraction variable is ubiquitously used in combustion modeling to describe the "mixedness" between fuel and oxidizer. The quantities $\rho_0$ and $\rho_1$ are the pure component densities, i.e. $\rho_0 = \rho(\phi = 0)$ and $\rho_1 = \rho(\phi = 1)$.

The first example problem is a one-dimensional manufactured solution reflective of binary diffusive mixing:

$$\begin{aligned}
\phi(x,t) &= \frac{\exp(-k_1 t) - \cosh(w_0 x \exp(-k_2 t))}{\exp(-k_1 t)\left(1 - \frac{\rho_0}{\rho_1}\right) - \cosh(w_0 x \exp(-k_2 t))} \\
\rho(x,t) &= \left(\frac{\phi(x,t)}{\rho_1} + \frac{1-\phi(x,t)}{\rho_0}\right)^{-1} \\
u(x,t) &= 2k_2 \exp(-k_1 t)\frac{\rho_0 - \rho_1}{\rho(x,t)}\left(\frac{\widehat{u}x}{\widehat{u}^2 + 1} + \frac{(\frac{k_1}{k_2} - 1)(\arctan\widehat{u} - \frac{\pi}{4})}{w_0 \exp(-k_2 t)}\right)
\end{aligned} \tag{4.2}$$

where $\widehat{u} = \exp(w_0 x \exp(-k_2 t))$ and $w_0, k_1$, and $k_2$ are constant parameters. Note that (4.2) satisfies the continuous continuity equation (2.3) with $\bar{\bar{S}}_{\bar{\rho}} = 0$, but produces a non-zero source term in the scalar transport equation (2.5). No source term is specified in the momentum equation, instead the pressure is allowed to compensate to satisfy (2.4) with $\bar{\bar{S}}_{\tilde{u}_i} = 0$. If interested, one could solve for the analytical pressure distribution by integrating (2.4) with respect to $x$. The relevant manufactured scalar source term is computed by substituting (4.2) into (2.5) and solving for $\bar{\bar{S}}_{\tilde{\phi}_k}$. The spatio-temporal evolution of (4.2) is shown in Figure 4.1 for the parameter values in Table 4.1. The computational domain for this problem is $0 \le x \le 2$

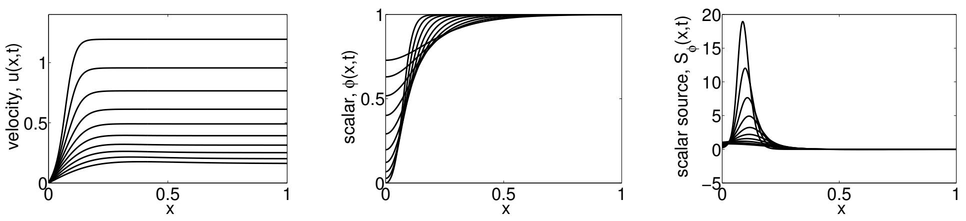


FIG. 4.1. *1-D manufactured solution (l to r): $u(x,t)$, $\phi(x,t)$, $\bar{\dot{S}}_\phi(x,t)$.*

TABLE 4.1
*Parameter values for 1-D problem.*

| parameter | value |
|---|---|
| $\rho_0$ | 20 |
| $\rho_1$ | 1 |
| $k_1$ | 4 |
| $k_2$ | 2 |
| $w_0$ | 50 |
| $\bar\rho\tilde\alpha_\phi = \bar\mu$ | 0.03 |

TABLE 4.2
*Parameter values for 2-D problem.*

| parameter | value | parameter | value |
|---|---|---|---|
| $\rho_0$ | 20 | $a$ | 1/5 |
| $\rho_1$ | 1 | $b$ | 20 |
| $k$ | 5 | $u_f$ | 1/4 |
| $\omega$ | 1 | $v_f$ | 0 |
| $\bar\rho\tilde\alpha_\phi$ | 0.001 | $\bar\mu$ | 0.001 |

and $0 \le t \le 1$. A similar problem was investigated in [25], although not within the framework of MMS.

A second MMS problem involves a two-dimensional corrugated front with advection and diffusion:

$$\begin{aligned}
\phi(x,y,t) &= \frac{1+\tanh(b\widehat{x}\exp(-\omega t))}{(1+\frac{\rho_0}{\rho_1}) + (1-\frac{\rho_0}{\rho_1})\tanh(b\widehat{x}\exp(-\omega t))} \\
\rho(x,y,t) &= \left(\frac{\phi(x,y,t)}{\rho_1} + \frac{1-\phi(x,y,t)}{\rho_0}\right)^{-1} \\
u(x,y,t) &= \frac{\rho_1-\rho_0}{\rho(x,y,t)}\Big(-\omega\widehat{x} + \frac{\omega\widehat{x}-u_f}{\exp(2b\widehat{x}\exp(-\omega t))+1} \\
&\qquad + \frac{\omega\log(\exp(2b\widehat{x}\exp(-\omega t))+1)}{2b\exp(-\omega t)}\Big) \\
v(x,y,t) &= v_f \\
p(x,y,t) &= 0
\end{aligned} \tag{4.3}$$

where $\widehat{x}(x,y,t) = u_f t - x + a\cos(k(v_f t - y))$ and $a, b, k, \omega, u_f$, and $v_f$ are constant parameters. Equation (4.3) satisfies the continuous continuity equation (2.3) with $\bar{\dot{S}}_{\bar\rho} = 0$. Non-zero source terms appear in the $x$ and $y$ momentum equations (2.4) and the scalar transport equation (2.5). The initial density field described by (4.3) is shown in Figure 4.2 given the parameter values in Table 4.2. The computational domain for this problem is $-1 \le x \le 2$, $-1/2 \le y \le 1/2$, and $0 \le t \le 1$.

In order to focus on the effects of density, the viscosity $\bar\mu$ and the "dynamic" diffusivity $\bar\rho\tilde\alpha_k$ in equations (2.4) and (2.5) are assumed constant in both the one- and two-dimensional MMS examples.

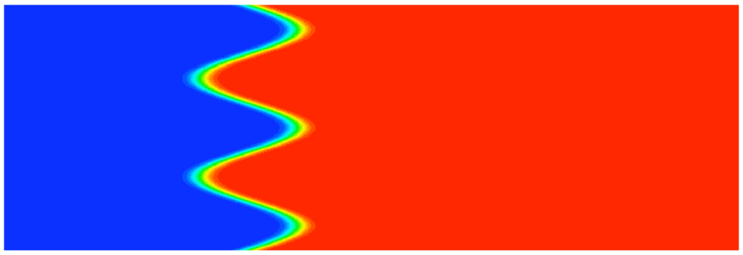
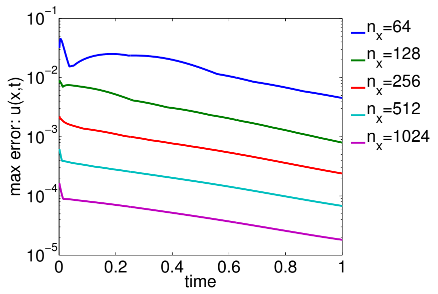


FIG. 4.2. *2-D manufactured solution:* $\rho(x, y, t)$ *at* $t = 0$ (*red:* $\rho = 20$, *blue:* $\rho = 1$).



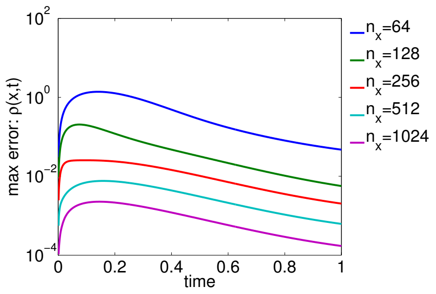


FIG. 5.1. *Maximum error in velocity* $u(x, t)$ *versus time for 1-D manufactured solution.*

FIG. 5.2. *Maximum error in density* $\rho(x, t)$ *versus time for 1-D manufactured solution.*

**5. Results.** A spatial grid-refinement study using the one-dimensional example problem (4.2) was conducted to assess the convergence properties of CDP's numerics. Computational grids consisting of 64, 128, 256, 512, and 1024 control volumes were used. A time step of $\Delta t = 0.00125$ was applied in all cases, leading to CFL numbers in the range 0.048 to 0.764. The boundary conditions at $x = 0$ were $u = 0$ and $\partial\phi/\partial x = 0$. An "outlet" boundary condition was applied at $x = 2$, a location sufficiently removed from the action so as to not to introduce significant errors in the solution. The velocity, pressure, and scalar values were solved from equations (2.3)-(2.5), and the density was evaluated using the analytic function (4.1) and the instantaneous scalar field. The convergence tolerance for solving transport equations and the pressure Poisson equation was $1 \times 10^{-8}$. Inner iterations at each time step were continued until the maximum density difference between iterations $|\rho^{m+1} - \rho^m|$ was less than $1 \times 10^{-8}$.

The maximum error ($L_\infty$-norm) in $u(x, t)$, $\phi(x, t)$, and $\rho(x, t)$ was monitored throughout the simulation and shows second-order convergence for all variables with respect to $\Delta x$. Detailed convergence results of the exercise are tabulated in Table 5.1. Plots of the maximum error versus time for $u(x, t)$ and $\rho(x, t)$ are shown in Figures 5.1 and 5.2. Note that the error smoothly decays with time in each simulation, as the flow features diffuse and become easier to resolve. As part of a separate temporal-refinement study (not shown here) the time step was halved to $\Delta t = 0.000625$ and the simulations were repeated on the $n_x = 1024$ grid. The results were almost indistinguishable, with a maximum difference on the order of $10^{-6}$. This suggests that the results are "converged" in a temporal sense, and that time errors are subservient to spatial errors at these resolutions.

In order to evaluate the effect of EOS tabulation on code performance, a refinement study was conducted in which the EOS (4.1) was interpolated linearly from successively-refined

Table 5.1
*1-D manufactured solution: spatial grid refinement.*

| no. of points | max error $u(x,t)$ | observed order | max error $\phi(x,t)$ | observed order | max error $\rho(x,t)$ | observed order |
|---|---|---|---|---|---|---|
| 64 | 4.4897e-02 | | 3.3918e-02 | | 1.3806e+00 | |
| 128 | 8.7491e-03 | 2.36 | 6.8977e-03 | 2.30 | 2.0429e-01 | 2.76 |
| 256 | 2.2050e-03 | 1.99 | 2.1180e-03 | 1.70 | 2.5377e-02 | 3.01 |
| 512 | 6.1929e-04 | 1.83 | 5.9749e-04 | 1.83 | 7.6128e-03 | 1.74 |
| 1024 | 1.6394e-04 | 1.92 | 1.5882e-04 | 1.91 | 2.2636e-03 | 1.75 |

Table 5.2
*Tabulated EOS resolutions and errors.*

| no. of points | max error $\rho(\phi)$ | avg error $\rho(\phi)$ |
|---|---|---|
| 21 | 1.6118e+00 | 7.3605e-02 |
| 31 | 9.4647e-01 | 3.3885e-02 |
| 51 | 4.4249e-01 | 1.2463e-02 |
| 101 | 1.3878e-01 | 3.1475e-03 |
| 201 | 3.9362e-02 | 7.8898e-04 |
| 401 | 1.0521e-02 | 1.9738e-04 |

grids of uniformly-spaced points in $\phi$-space. A summary of the tabulation resolutions and their associated errors can be found in Table 5.2. Interpolation of the EOS at the coarsest resolutions in Table 5.2 would not be unreasonable in many engineering calculations where property tables are multi-dimensional (typically 3-4) and memory is limited.

The simulations were effected on a grid of 1024 control volumes with a time step of $\Delta t = 0.00125$. The boundary conditions and solver convergence limits were identical to the spatial grid-refinement study above. The full convergence results are tabulated in Table 5.3, and plots of the temporal evolution of the error for $u(x,t)$ and $\rho(x,t)$ are shown in Figures 5.3 and 5.4. The "$n_\phi = \infty$" label indicates results using the analytic or non-interpolated EOS.

The data clearly indicate a degradation of accuracy when using a tabulated EOS. Velocity convergence is tending towards first-order behavior, while scalar and density convergence appears to be closer to second-order (with respect to $\Delta\phi$). These trends, however, are speculative at best as the data are not well converged, even with 401 interpolation points in the EOS. It is likely that scalar convergence outperforms velocity because of the manufactured source term in (2.5). In the simulations, the scalar source was evaluated as a function of $x$ and $t$, rather than $u(x,t)$, $\phi(x,t)$, and $\rho(x,t)$. The source term, therefore, implicitly used the analytic EOS and was partially shielded from the influence of tabulation errors. It is not surprising, therefore, that scalar convergence was less affected than velocity, especially when considering the relative strength of the scalar source term in this example (see Figure 4.1).

In addition to poor convergence rates, it is clear that EOS interpolation errors dramatically affect the character of the error in the field variables. The smooth error decay exhibited in Figure 5.1 is replaced by the unsteadiness apparent in Figure 5.3. These numerical fluctuations result from the tight coupling between density, velocity, and pressure in low-Mach number projection methods. Density errors that arise from the tabulation, are readily translated into velocity errors as the pressure acts to "correct" changes in the global mass-balance. The velocity and density in turn influence the evolution of the scalar field in a non-linear manner, adding further complexity. The end result is that small errors in the EOS evaluation

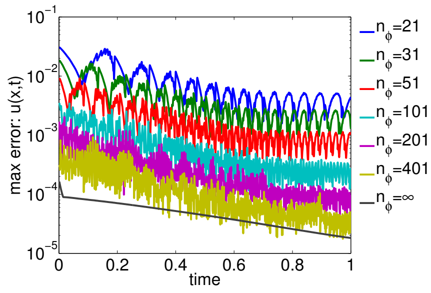


FIG. 5.3. *Maximum error in velocity $u(x, t)$ versus time for 1-D manufactured solution.*

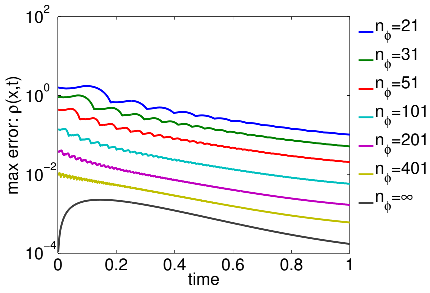


FIG. 5.4. *Maximum error in density $\rho(x, t)$ versus time for 1-D manufactured solution.*

TABLE 5.3
*1-D manufactured solution: EOS look-up table refinement.*

| no. of points | max error $u(x,t)$ | observed order | max error $\phi(x,t)$ | observed order | max error $\rho(x,t)$ | observed order |
|---|---|---|---|---|---|---|
| 21 | 3.0450e-02 | | 5.4061e-02 | | 1.7409e+00 | |
| 31 | 1.8274e-02 | 1.31 | 2.8370e-02 | 1.66 | 1.0069e+00 | 1.41 |
| 51 | 9.2087e-03 | 1.38 | 1.1613e-02 | 1.79 | 4.6242e-01 | 1.56 |
| 101 | 3.5173e-03 | 1.41 | 3.2998e-03 | 1.84 | 1.4308e-01 | 1.72 |
| 201 | 2.0185e-03 | 0.81 | 9.6504e-04 | 1.79 | 4.0427e-02 | 1.84 |
| 401 | 9.2466e-04 | 1.13 | 3.4493e-04 | 1.49 | 1.0764e-02 | 1.92 |
| ∞ | 1.6394e-04 | | 1.5882e-04 | | 2.2636e-03 | |

can amplify and lead to large errors in the velocity and scalar fields.

The impact of a tabulated EOS has likewise been studied in the context of the two-dimensional MMS problem (4.3). The full study is still in progress, however, preliminary results are reported here. The computational grid comprises 600 × 200 hexahedral control volumes and a uniform time step of $\Delta t = 0.005$. Dirichlet boundary conditions were imposed at $x = -1$ and $y = \pm 1/2$, and an "outlet" boundary condition was applied at $x = 2$. All solvers and inner iterations were converged to a tolerance of $1 \times 10^{-6}$ at all time steps and in all simulations.

The time-evolution of the error is depicted in Figure 5.5. It should be noted that the analytic-EOS solution ($n_\phi = \infty$), appears to still contain significant spatial error and should be further refined. Despite this, definite trends are evident as the EOS grid is refined. First, it appears that all variables converge to the $n_\phi = \infty$ solution when the EOS is sufficiently refined. Second, velocity and pressure appear to be much more prone to spurious fluctuations than density or its underlying scalar (not shown). The fluctuations tend to persist even for very fine EOS grids. Third, for coarse interpolations of the EOS, errors tend to compound with time, suggesting that uncontrolled EOS errors can have a dramatic effect on the temporal development of the flow. Indeed, Figure 5.6 chronicles a steady departure of the numerical solution from the manufactured solution when the EOS is coarsely interpolated. The numerical fluctuations induced by EOS interpolation errors undoubtedly find expression in the flow variables on a macro-scale. This is readily visible in Figure 5.7, which shows the convective outlet velocity, $u(x = 2)$ for different EOS resolutions. Here, interpolation errors cause dramatic fluctuations about the exact value of 0.2375. The presence of these fluctuations, whose genesis is entirely numerical, holds serious implications for subgrid modeling of combustion and turbulence phenomena.

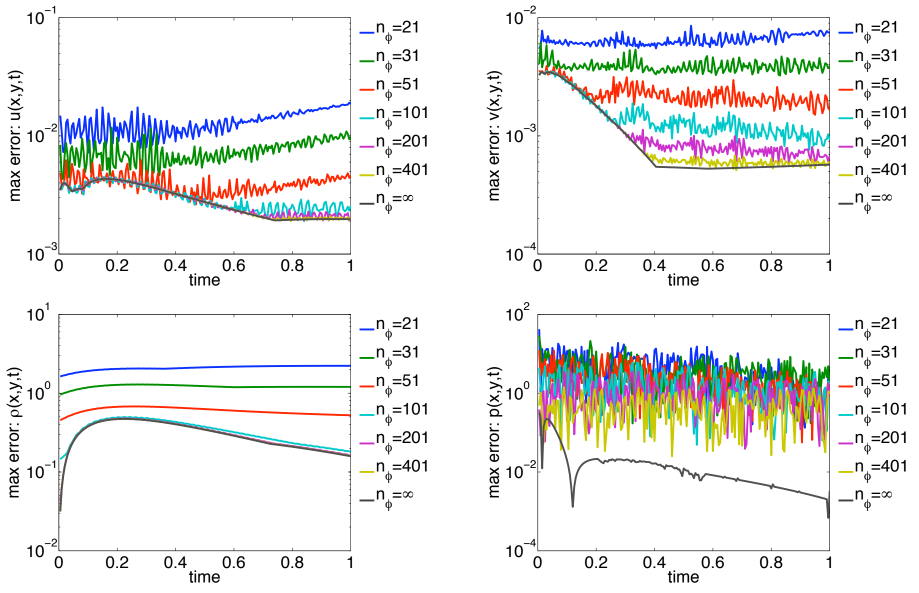


FIG. 5.5. *2-D manufactured solution, maximum error.* (*clockwise from top left*) $u(x, y, t)$, $v(x, y, t)$, $p(x, y, t)$, $\rho(x, y, t)$.

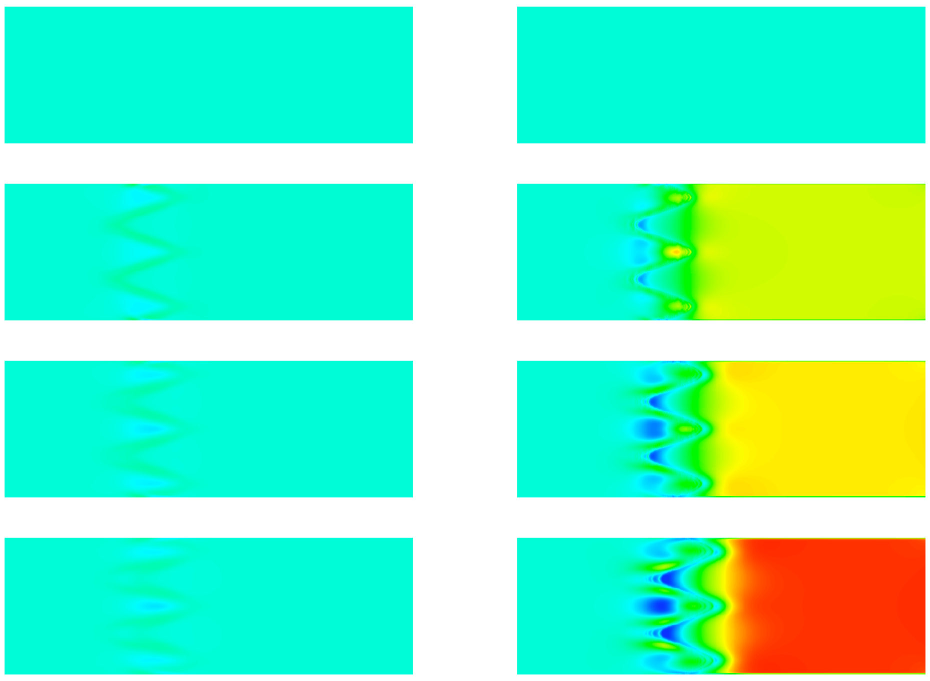


FIG. 5.6. *2-D manufactured solution, error in* $u(x, y, t)$. *The left figures use analytic EOS-evaluations, and the right figures use linearly-interpolated EOS-evaluations with* $n_\phi = 21$. (*top to bottom*) $t = 0$, $t = 0.33$, $t = 0.67$, $t = 1$. (*red:* $u_{err} = 0.019$, *blue:* $u_{err} = -0.008$ )

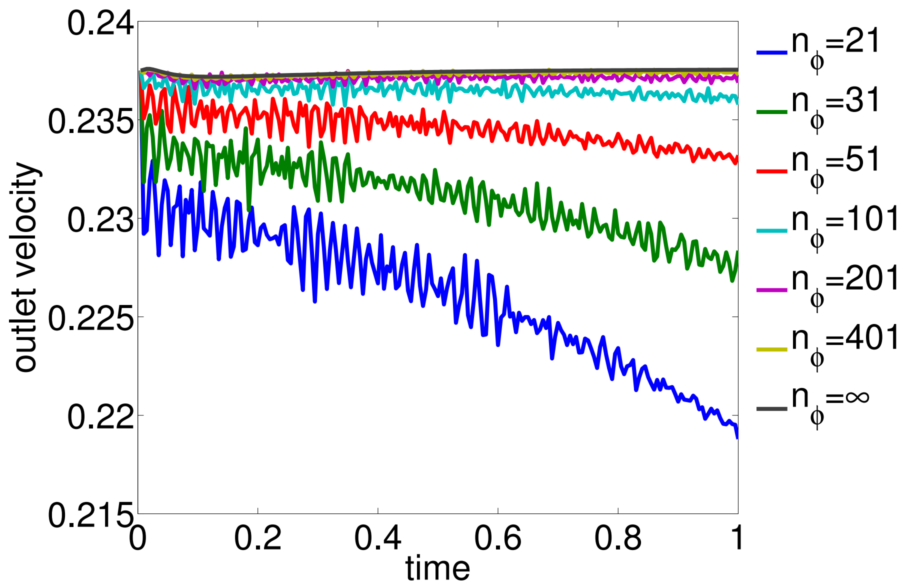


FIG. 5.7. *2-D manufactured solution: convective outlet velocity* $u(x, y, t)$ *at* $x = 2$.

**6. Conclusions.** In this study, the Method of Manufactured Solutions was used to investigate the effects of tabulated state-equations on the convergence and accuracy of the multiphysics hydrodynamics code CDP. Two MMS problems were constructed whose evolution is reflective of some of the basic physics germane to combustion problems, namely: diffusive mixing of species and convection of density fronts. Both of the MMS examples analytically satisfy the source-free continuity equation, and use manufactured source terms to balance the momentum and scalar transport equations. Grid refinement studies performed using the MMS problems confirm the spatial convergence rate of CDP to be second-order when an analytic EOS was used. Convergence of the flow variables to the exact solution was markedly impaired when the EOS was linearly interpolated in $\phi$-space. The MMS results indicate that EOS interpolation errors introduce spurious numerical fluctuations in the flow variables, with velocity and pressure being particularly vulnerable. These errors tend to accumulate with time and can potentially alter the temporal evolution of the solution in variable-density flows.

## REFERENCES

[1] R. B. BOND, C. C. OBER, AND P. M. KNUPP, *A manufactured solution for verifying CFD boundary conditions, part III*, in 36th AIAA Fluid Dynamics Conference, vol. 3, San Francisco, CA, June 2006, pp. 1966–1982.

[2] T. A. BRUNNER, *Development of a grey nonlinear thermal radiation diffusion verification problem*, Transactions of the American Nuclear Society, 95 (2006), pp. 876–878.

[3] S. P. DOMINO, G. WAGNER, A. LUKETA-HANLIN, A. BLACK, AND J. SUTHERLAND, *Verification for multi-mechanics applications*, in 48th AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference, Honolulu, HI, April 2007.

[4] L. ECA, M. HOEKSTRA, A. HAY, AND D. PELLETIER, *On the construction of manufactured solutions for one- and two-equation eddy-viscosity models*, Int. J. Num. Meth. Fluids, 54 (2007), pp. 119–154.

[5] R. D. FALGOUT AND U. M. YANG, *HYPRE: a library of high performance preconditioners*, in Computational Science - ICCS 2002 Part III, P. M. A. Sloot, C. J. K. Tan, J. J. Dongarra, and A. G. Hoekstra, eds., vol. 2331 of Lecture Notes in Computer Science, Springer-Verlag, 2002, pp. 632–641.

[6] M. Germano, U. Piomelli, P. Moin, and W. H. Cabot, *A dynamic subgrid-scale eddy viscosity model*, Phys. Fluids A, 3 (1991), pp. 1760–1765.

[7] S. Ghosal, T. S. Lund, P. Moin, and K. Akselvoll, *A dynamic localization model for large-eddy simulation of turbulent flows*, J. Fluid Mech., 286 (1995), pp. 229–255.

[8] F. Ham, K. Mattsson, and G. Iaccarino, *Accurate and stable finite volume operators for unstructured flow solvers*, Annual Research Briefs 2006, Center for Turbulence Research, Stanford University, NASA Ames (2006), pp. 243–261.

[9] V. E. Henson and U. M. Yang, *BoomerAMG: a parallel algebraic multigrid solver and preconditioner*, Appl. Num. Math., 41 (2002), pp. 155–177.

[10] A. Y. Klimenko and R. W. Bilger, *Conditional moment closure for turbulent combustion*, Prog. Energy Combust. Sci., 25 (1999), pp. 595–687.

[11] P. Knupp and K. Salari, *Verification of Computer Codes in Computational Science and Engineering*, Chapman & Hall/CRC, Boca Raton, 2003.

[12] P. Moin, K. Squires, W. Cabot, and S. Lee, *A dynamic subgrid-scale model for compressible turbulence and scalar transport*, Phys. Fluids A, 3 (1991), pp. 2746–2757.

[13] W. L. Oberkampf, T. G. Trucano, and C. Hirsch, *Verification, validation, and predictive capability in computational engineering and physics*, Appl. Mech. Rev., 57 (2004), pp. 345–384.

[14] N. Peters, *Laminar diffusion flamelet models in non-premixed turbulent combustion*, Prog. Energy Combust. Sci., 10 (1984), pp. 319–339.

[15] ———, *Turbulent Combustion*, Cambridge University Press, Cambridge, 2000.

[16] C. D. Pierce and P. Moin, *A dynamic model for subgrid-scale variance and dissipation rate of a conserved scalar*, Phys. Fluids, 10 (1998), pp. 3041–3044.

[17] ———, *Progress-variable approach for large-eddy simulation of turbulent combustion*, PhD thesis, Stanford University, 2001.

[18] ———, *Progress-variable approach for large-eddy simulation of non-premixed turbulent combustion*, J. Fluid Mech., 504 (2004), pp. 73–97.

[19] S. B. Pope, *PDF methods in turbulent reactive flows*, Prog. Energy Combust. Sci., 11 (1985), pp. 119–192.

[20] P. J. Roache, *Verification and Validation in Computational Science and Engineering*, Hermosa Publishers, Albuquerque, 1998.

[21] ———, *Verification of codes and calculations*, AIAA Journal, 36 (1998), pp. 696–702.

[22] ———, *Code verification by the method of manufactured solutions*, J. Fluids Eng., 124 (2002), pp. 4–10.

[23] C. J. Roy, *Review of code and solution verification procedures for computational simulation*, J. Comp. Phys., 205 (2005), pp. 131–156.

[24] C. J. Roy, C. C. Nelson, T. M. Smith, and C. C. Ober, *Verification of Euler/Navier-Stokes codes using the method of manufactured solutions*, Int. J. Num. Meth. Fluids, 44 (2004), pp. 599–620.

[25] L. Shunn and F. Ham, *Consistent and accurate state evaluations in variable-density flow simulations*, Annual Research Briefs 2006, Center for Turbulence Research, Stanford University, NASA Ames (2006), pp. 135–147.

[26] J. Smagorinsky, *General circulation experiments with the primitive equations*, Mon. Weather Rev., 91 (1963), pp. 99–164.

[27] D. Tremblay, S. Etienne, and D. Pelletier, *Code verification and the method of manufactured solutions for fluid-structure interaction problems*, in 36th AIAA Fluid Dynamics Conference, vol. 2, San Francisco, CA, June 2006, pp. 882–892.

# VALIDATION OF ANALYTICAL MODELS FOR FAULT TOLERANCE

MARIA RUIZ VARELA[†] AND RON A. OLDFIELD[‡]

**Abstract.** Traditional application-directed checkpointing approaches, in particular checkpoint-to-disk, might not scale for next-generation massively parallel processing (MPP) systems. Analytical modeling of the checkpointing overhead on such massive-scale systems supports this claim. These analytical models provide valuable insight of the behavior of checkpointing; however, there are aspects of the checkpoint operation that are complex to model mathematically; thus, the next step to obtain accurate figures of the impact of checkpointing on the performance of large-scale applications is to validate these analytical models on a massive-scale supercomputer. This work describes our efforts to develop software to validate the analytical model of checkpoints on the Red Storm supercomputer at Sandia National Laboratories.

**1. Introduction.** A modern massively parallel processing (MPP) system, typically comprising tens of thousands of processors, executes applications that, by design, use a large number of these processors and can take days and even months to complete. Moreover, these large-scale applications address relevant scientific problems. Because of the size and the critical nature of these applications, adequate fault tolerance mechanisms need to be in place to enable scientists to obtain significant results in a reasonable amount of time. In case the system fails, checkpointing enables the application to restart from the immediate previous checkpoint, not from the beginning; saving computation time and computation resources.

Traditional checkpointing mechanisms have provided acceptable fault-tolerance for modern MPP systems. The question is if, given the predicted size of future systems, will current mechanisms, specifically checkpointing-to-disk approaches, scale for next-generation MPP systems consisting of hundreds of thousands of processors. Checkpoint-to-disk is currently the most widely used method in massive-scale systems. As the number of processors in MPP systems increases, the use of traditional checkpoint-to-disk as fault tolerance mechanisms is likely to cause the storage system to become a bottleneck.

Previous collaborative work between Sandia National Laboratories (SNL) and The University of Texas at El Paso (UTEP) produced analytical models of the impact of checkpoints on the performance of applications executing on next-generation MPP systems. These models showed that, as the size of the system increases, current application-directed, checkpointing-to-disk strategies will negatively affect application performance, where checkpointing can take as much as 50 percent of the application's total execution time. This predicted high checkpoint overhead makes current approaches unsuitable fault tolerance mechanisms for future systems.

Although analytical models provided valuable insight into the behavior of the checkpoint operation, it is difficult to mathematically model it in its entirety. Because of the inherent complexity of mathematical modeling, to obtain analytical models, it is necessary to make several simplifying assumptions; for instance, the analytical models of the checkpoint operation assume that there is no network or storage contention, that all processors write the same amount of state and that the parallel system is perfectly scalable. Failures in the overlay network or the storage system are not modeled. In most cases, these assumptions are not in accordance with the actual behavior of real systems. Consequently, to obtain accurate figures on the impact of checkpoints on application performance, it is imperative to validate the analytical models on a large-scale system. Experimental validation on a large-scale system can provide valuable information about system behavior that is complex to model mathematically, e.g., overheads associated with the network and the storage system.

[†]The University of Texas at El Paso, mdruizvarela@miners.utep.edu
[‡]Sandia National Laboratories, raoldfi@sandia.gov

Table 2.1
*MPP systems parameter values*

| Parameter | Red Storm | BlueGene/L | Jaguar | Petaflop |
|---|---|---|---|---|
| $n_{max} \times cores$ | $12,960 \times 2$ | $65,536 \times 2$ | $11,590 \times 2$ | $50,000 \times 2$ |
| $d_{max}$ | 1GB | 0.25GB | 2.0GB | 2.5GB |
| $\beta_s$ | 50GB/s | 45GB/s | 45GB/s | 500GB/s |
| $\beta_n$ | 2.3TB/s | 360GB/s | 1.8TB/s | 30TB/s |
| $\beta_L$ | 4.8GB/s | 1.4GB/s | 3.8GB/s | 40GB/s |

In this context, the objective of this work is to develop software to validate the analytical models of the checkpoint operation on the Red Storm supercomputer at SNL.

**2. Background.** Reducing the checkpoint overhead is essential to improve application performance. Checkpoint overhead as defined in [11], is the increase in total application execution time caused by checkpointing. There are several techniques that have been proposed to ameliorate the effect of checkpoints on application performance. The work in [5] proposes a checkpoint mechanism that combines a lightweight file system (LWFS) and an overlay network to reduce checkpoint overhead, and consequently, improve application performance.

An overlay network can be tough as a virtual topology that is built on top of a physical underlying network or protocol [4]. In the LWFS+overlay approach, each compute node writes its state to intermediate nodes of the overlay network. In a traditional checkpoint-to-disk approach [9], application state is written directly to stable storage, where stable storage is persistent, rather than volatile, as in checkpoint-to-memory approaches [10] [2]. The intermediate nodes of the overlay network enable the buffering of checkpoint data to disk. This strategy enables the application to transfer checkpoint data at speeds governed by the network bandwidth rather than by the storage bandwidth. The use of an overlay network in this manner helps prevent the I/O storage system from becoming a bottleneck, improving I/O performance of checkpointing. Table 2.1 shows different parameters for several massive-scale, in-production systems and a theoretical Petaflop system. It can be seen across systems that the difference between storage and network bandwidths can be as large as one order of magnitude.

Once in the intermediate nodes, instead of being saved to a traditional file system, checkpoint data is saved to the LWFS to speed up the time it takes to save the checkpoint file to persistent storage. LWFS follows the lightweight approach used in the design of Catamount, the operating system of the Red Storm supercomputer [1]. The compute nodes in Red Storm use a lightweight operating system that does not support operations such as threading, multitasking, or memory management. Other nodes such as I/O nodes and service nodes use a more traditional or heavyweight operating system to provide shared services, e.g. Linux. Following this philosophy, the LWFS core supports only essential operations, which are defined as those operations that do not prevent or degrade application scalability. Other non-essential functionality is left to the application to implement it [7] [8]. The main advantage of the lightweight storage architecture compared to a traditional file system is that it provides fast and direct access to storage. These features make the use of lightweight storage architectures suitable for checkpointing. In a traditional file system, the I/O requests made by diskless compute nodes of an MPP system send have to be sent through the network. LWFS provides a mechanism to enable the clients, i.e., the overlay nodes that needs to save checkpoint data, to directly access the storage devices. Figure 2.1 shows the LWFS core architecture.

The analytical models of the checkpoint operation provided in [6] showed that application-directed checkpoint-to-disk will likely cause the storage system to become a bot-

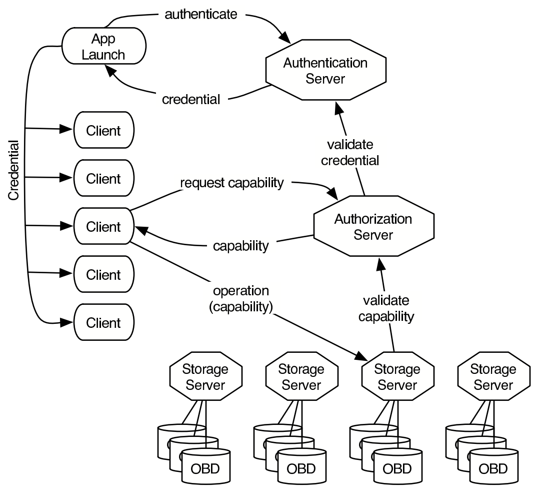


FIG. 2.1. *The LWFS core architecture in [7]*

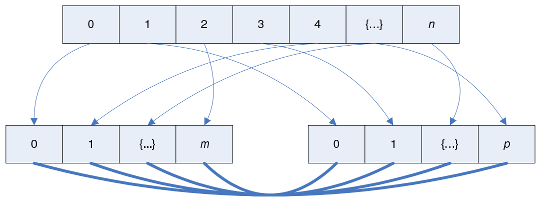


FIG. 2.2. *Creation of an intracommunicator [3]. The original group of n processes,* MPI_COMM_WORLD, *is partitioned into two groups, one for the compute processes and other for the overlay processes. An intercommunicator between the two is created.*

tleneck for next-generation MPP systems. LWFS+overlay checkpointing seeks to alleviate this problem. To model the use of the overlay network during the checkpointing operation, the analytical model incorporates the bandwidth of the checkpoint operation as a parameter for describing the time required to write a checkpoint file to stable storage:

$$\delta = \alpha_c + \frac{nd}{\beta_{chkpt}},$$

where:

| | | |
|---|---|---|
| $\alpha_c$ | = | Start-up cost of a checkpoint operation, |
| $n$ | = | Number of processors used by the application, |
| $d$ | = | Amount of data written by each processor, |
| $\beta_{chkpt}$ | = | Perceived bandwidth of a checkpoint operation, i.e., $\min(n\beta_L, \beta_n, \beta_s)$, |
| $\beta_L$ | = | One-way network bandwidth per link, |
| $\beta_n$ | = | Aggregate bisection network bandwidth, and |
| $\beta_s$ | = | Aggregate storage system bandwidth. |

The results in [6] showed that when checkpoint data fits in the intermediate nodes, the LWFS+overlay approach outperforms traditional checkpoint-to-disk.

**3. Method.** A proof of concept application was implemented to validate the analytical models developed in [7]. This MPI application creates compute processes that perform useful computation work and, after a fixed period of time, pauses to checkpoint its computation state. For our purposes, the compute nodes do not perform any useful computation, they remain idle during the checkpointing period. The duration of the computation period is determined by the value of the checkpoint interval, which is the application execution time elapsed between two consecutive checkpoint operations.

**3.1. The overlay network.** In the LWFS+overlay checkpointing approach, the compute nodes of the MPP system save computation state to intermediate nodes in the overlay network, rather than saving it directly to storage. To simulate the overlay network, the application uses communicators to create a subset of processes from the original communicator. When MPI is initialized, a set of processes associated with the default communicator, `MPI_COMM_WORLD`, is created. `MPI_Comm_split` creates a new subset of processes from the original set of processes associated with `MPI_COMM_WORLD`. This new intracommunicator comprises the overlay processes. The number of overlay processes can be set on an input configuration file that is read by the MPI application. `MPI_COMM_WORLD` is a partition of two disjoint sets, the set of the compute processes, and the set of the overlay nodes. To link processes of these two groups, `MPI_Intercomm_create` creates a new intercommunicator from two existing intracommunicators. This enables communication between the group of compute processes and the group of overlay processes. Figure 2.2 shows the creation of the intercommunicator.

**3.2. The LWFS programming interface.** The LWFS programming interface provides services that enable clients to write directly to storage. In the LWFS+overlay checkpointing approach, the compute nodes save the state to intermediate nodes of the overlay network. The application stores pending write requests in a linked list data structure. Depending on the file system configuration option, checkpoint files are written to a Unix file system or to the LWFS. In case the application is configured to save to the LWFS, it initializes the data structures used by the LWFS programming interface. The LWFS initialization:

1. parses the configuration file,
2. gets the services descriptions for the services, i.e. the authorization and storage services,
3. broadcasts them to all processes.

The application creates a container to which processes will save objects. To write to the container it is required to first obtain the capability that enables the client process to write objects to that container. The capability is then broadcasted to all the overlay processes. Since checkpointing only requires to write to the LWFS, overlay processes use the same container ID to save checkpoint files. When the application execution pauses to perform a checkpoint, each process creates a checkpoint file in parallel. In this file-per-process operation, each process that needs to save a file initializes and creates the object structure that stores the data that will be written to the LWFS. After the object structure is created, the file is saved to storage.

**4. Conclusions.** We described a proof of concept application that simulates a parallel application that checkpoints using the LWFS+overlay approach. This work is a preliminary effort oriented at developing software to validate the analytical models of the checkpoint operation proposed in [7]. While analytical modeling provided valuable insight of the checkpoint operation, it is complex to model all of its aspects; thus, experimental validation can help to better understand the effect of checkpointing on modern and next-generation MPP systems. It is planned that the development efforts will continue and that the latest version of the validation software will be executed on the Red Storm supercomputer at SNL.

REFERENCES

[1] W. J. Camp and J. L. Tomkins, *The Red Storm computer architecture and its implementation*, in The Conference on High-Speed Computing: LANL/LLNL/SNL, Salishan Lodge, Glenedon Beach, Oregon, 2003.

[2] C. Engelmann and A. Geist, *A diskless checkpointing algorithm for super-scale architectures applied to the fast fourier transform*, in Proceedings of the International Workshop on Challenges of Large Applications in Distributed Environments, Seattle, WA, June 2003, IEEE Computer Society Press, pp. 47–52.

[3] I. Foster, *Designing and Building Parallel Programs: Concepts and Tools for Parallel Software Engineering*, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995.

[4] A. Gavrilovska, K. Schwan, O. Nordstrom, and H. Seifu, *Network processors as building blocks in overlay networks*, in Proceedings of the 11 th Symposium on High Performance Interconnects (HOTI03), August 2003, pp. 83–88.

[5] R. Oldfield, *Investigating lightweight storage and overlay networks for fault tolerence*, in Proceedings of the High Availability and Performance Computing Workshop, Santa Fe, NM, October 2006.

[6] R. Oldfield, S. Arunagiri, P. Teller, S. Seelam, M. Varela, R. Riesen, and P. Roth, *Modeling the impact of checkpoints on next-generation systems*, in Proceedings of the IEEE Conference on Mass Storage Systems and Technologies (to appear), San Diego, CA, September 2007.

[7] R. Oldfield, A. Maccabe, S. Arunagiri, T. Kordenbrock, R. Riesen, L. Ward, and P. Widener, *Lightweight I/O for scientific applications*, in Proceedings of the IEEE International Conference on Cluster Computing, Barcelona, Spain, September 2006.

[8] R. Oldfield, P. Widener, A. Maccabe, L. Ward, and T. Kordenbrock, *Efficient data-movement for lightweight I/O,*, in Proceedings of the IEEE International Conference on Cluster Computing, Barcelona, Spain, September 2006, pp. 1–9.

[9] J. S. Plank, *Improving the performance of coordinated checkpointers on networks of workstations using RAID techniques*, in Proceedings of the Symposium on Reliable Distributed Systems, 1996, pp. 76–85.

[10] L. M. Silva and J. G. Silva, *An experimental study about diskless checkpointing.*, in Proceedings of the 24th EUROMICRO Conference, Vasteras, Sweden, August 1998, IEEE Computer Society Press, pp. 395–402.

[11] N. H. Vaidya, *Impact of checkpoint latency on overhead ratio of a checkpointing scheme*, IEEE Transactions on Computers, 46 (1997), pp. 942–947.

# AN OPTIMIZATION UNDER UNCERTAINTY (OUU) ALGORITHM WITH AN EXAMPLE FROM DESIGN

FRANCESCA D. REALE-LEVIS[*], VICENTE J. ROMERO[†], AND LAURA P. SWILER[‡]

**Abstract.** We present an algorithm that has been designed to reduce the number of function evaluations required to perform optimization under uncertainty (OUU). This method minimizes variance while ensuring that a prescribed mean target level of system output is achieved. We apply the optimization procedure to an automotive device design robustness problem. We consider a snap-fit composed of a moving part and a stationary part which are designed to the specification of three uncertain design variables. The three design variables and an additional noise variable determine the friction force between the two parts. All four factors contribute to variability of the friction force. Using our process, we find a point in the design space of minimum variance subject to a constraint that the mean of the friction force is a set level. We consider an exhaustive search and reliability methods for comparison of results and cost.

**1. Introduction.** Optimization under uncertainty refers to performing an optimization procedure when there is uncertainty or noise in the variables to be optimized over. This algorithm has been designed to minimize the number of function evaluations necessary to perform optimization under uncertainty. We present a method that simultaneously minimizes the variance of a desired output response while maintaining the constraint that the mean is fixed at a certain level. We demonstrate the process using an automotive device design example. The optimization procedure incorporates robustness into the design. We find a set of values for the design variables that ensure repeatability throughout the manufacturing process and service life.

**2. A Taguchi Snap-Fit Example.** We demonstrate the process using an automotive device design robustness example. The optimization procedure incorporates repeatability into the target design leading to efficiency in the manufacturing process and improved satisfaction for the duration of customer use. The type of snap-fit we present is used to connect an encapsulated motor stator and motor housing [7].

Consider a snap-fit design [12] that is composed of both a moving part and a stationary part that are designed to the specification of three uncertain design variables: spring constant $K$ (N/mm), interference $I$ (mm), and ramp angle $\theta$ (degrees). A fourth noise variable is the friction coefficient $\mu$. The interference $I$ represents the vertical distance between the tip on the fixed part and the bottom edge of the tip on the moving part. In order to snap the parts together, the moving part must move past the tip of the fixed part. Whether or not the parts permanently snap together is largely based on the ramp angle. The setup is demonstrated in Figure 2.1

All four variables help to determine the friction force $Y$ (N) between the moving and stationary parts. This relationship is

$$Y = KI\frac{\mu + \tan\theta}{1 - \mu\tan\theta}. \tag{2.1}$$

The bounds on the nominal values of the design variables are 500 N/mm $\leq K \leq$ 600 N/mm, 0.1 mm $\leq I \leq$ 0.35 mm, and 45 degrees $\leq \theta \leq$ 65 degrees. We refer to these intervals as the design ranges of the design variables. The average value of the friction coefficient $\mu$ is 0.17. As specified in [12], all four variables are normally distributed with respective standard

[*]North Carolina State University, fdreale@ncsu.edu
[†]Sandia National Laboratories, vjromer@sandia.gov
[‡]Sandia National Laboratories, lpswile@sandia.gov

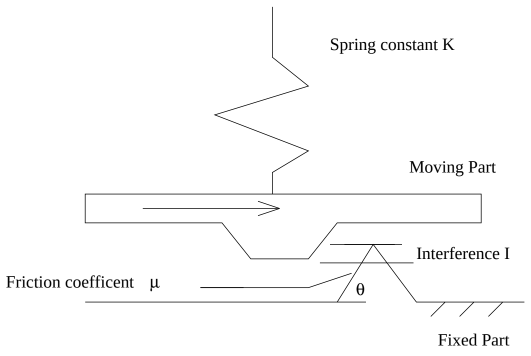


FIG. 2.1. *A snap-fit design example. Figure redrawn from Ref. [12].*

deviations $\sigma_K = 10$ N/mm, $\sigma_I = 0.017$ mm, $\sigma_\theta = 1$ degrees, and $\sigma_\mu = 0.017$. We assume that the above standard deviations hold for all nominal values of the four variables.

We aim to minimize variance and satisfy the constraint on the mean $\bar{Y} = 120$ N simultaneously.

**3. Reduction to Two Design Variables.** There are three design variables $K$, $I$, and $\theta$ which specify how the moving and stationary parts are related. There are four uncertain variables $K$, $I$, $\theta$, and $\mu$ which lead to variability in the result $Y$.

Let us analyze the three design variables and their effect on $Y$. We vary the nominal value of each variable while the nominal value of each other design variable is fixed at the center of its design range. The value of $\mu$ is fixed at its mean. Figure 3.1 demonstrates that over the design range of $K$ there is smaller change in $Y$ than there is over the design ranges of $I$ and $\theta$.

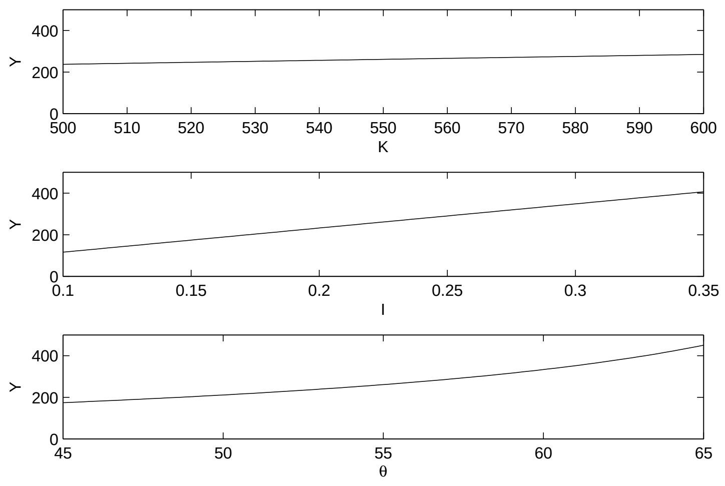


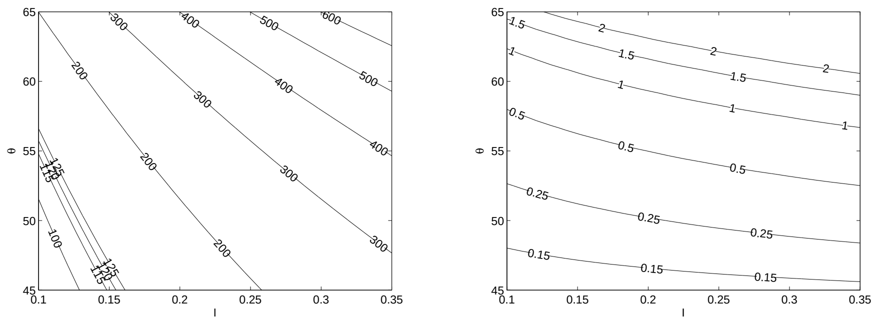
FIG. 3.1. *The friction force over the design range of each design variable.*

We reduce to two design variables so that we can better demonstrate the following robust optimization algorithm visually. Based on the relative importance of the three design variables, we relegate $K$ to be strictly a noise variable. Now there are two design variables $I$ and $\theta$ for which we change the nominal values throughout the optimization procedure. There are still four uncertain variables $K$, $I$, $\theta$, and $\mu$. The nominal value of $K$ is fixed at 550 N/mm and $\mu$ is fixed at 0.17 throughout the procedure.

**4. Analysis of the 2D Design Space.** The problem statement has slightly changed. Now, we optimize over two variables $I$ and $\theta$. We still aim to minimize variance and satisfy the constraint on the mean $\bar{Y} = 120$ N. We study the behavior of the friction force, its mean, and its variance.

We allow the two design variables to vary over their design ranges while we fix the nominal values of the other variables at $K = 550$ N/mm and $\mu = 0.17$. We use a $25 \times 25$ linearly spaced grid of points in the two dimensional design space. The mean and standard deviation are determined using 200 samples at each point.

We examine the design space for the contour $Y = 120$ N. Figure 4.1(a) displays the contours of $Y$. We find that the friction force is mildly nonlinear. During the process, we depend on the $Y = 120$ N contour being "close" to the $\bar{Y} = 120$ N contour. Figure 4.1(b) illustrates $\bar{Y} - Y$. The absolute difference between $Y$ and $\bar{Y}$ is less than one percent of the friction force throughout the design space.



(a) Contour plot of the friction force $Y$.  (b) Contour plot of $|\ \bar{Y} - Y\ |$.

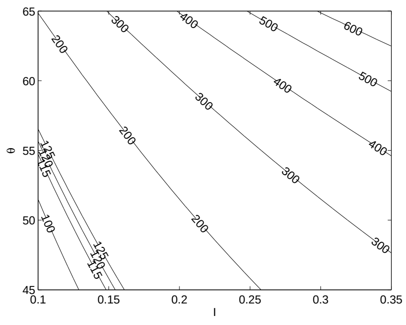
FIG. 4.1. *The friction force Y as well as the relationship between Y and the mean $\bar{Y}$.*

We study the contours for both the mean and variance. Figure 4.2(a). presents the mean contours. Figure 4.2(b). shows an approximation to the standard deviation contours. Based on the direction of decreasing standard deviation, we expect that the best point is at the lower, right end of the $\bar{Y} = 120$ N contour.

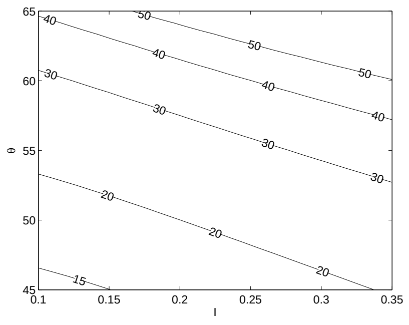
By relegating $K$ to be strictly a noise variable, we change the original problem. As we see from these plots, this is a mildly nonlinear function that we can visually track as well. Some of the techniques used in the following iterative process can be applied to an array of problems, some will not be as effective when applied to highly nonlinear problems.

**5. An iterative process: lowering variation while $Y$ = 120 N.** We employ an algorithm for finding an "optimal" point which has minimum variance while maintaining a friction force value of approximately 120 N. We first find the contour line $Y = 120$ N, then step along the contour to minimize variance.

For a visual representation of the process, see Figure 5.1 which provides the position of each iterate as well as the two tangent line approximations to $Y = 120$ N.



(a) A mean approximation using 200 random samples.



(b) A standard deviation approximation using 200 random samples.

FIG. 4.2. *The contours of the two-hundred sample mean and standard deviation.*

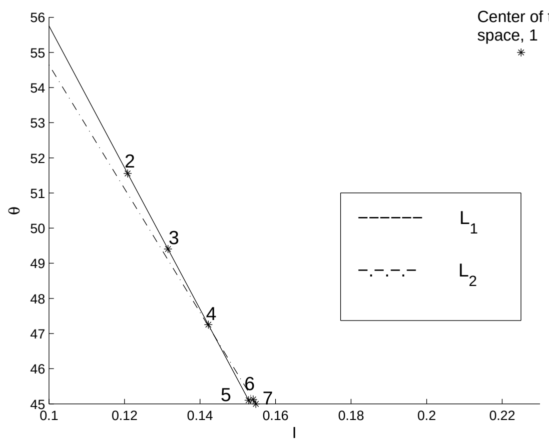


FIG. 5.1. *The iteration history and the approximations to $Y = 120$ N.*

**5.1. Finding $Y = 120$ N.** We maintain that two of the nominal values are fixed at $K = 550$ N/mm and $\mu = 0.17$. Consider a starting point at the center of the design space where $I = 0.225$ mm and $\theta = 55$ degrees.

**5.1.1. An Analytic Line Search Coefficient Approach.** We consider $F$ a function of $n$ variables where $F(\vec{x}) = F((x_1, x_2, ..., x_n))$. Suppose that we want to reach a target value of F which we call $F_t$.

The tangent hyperplane formula is

$$(F(\vec{x}) - F(\vec{x}_0)) = \frac{\partial F}{\partial x_1}(x_1 - [x_1]_0) + \ldots + \frac{\partial F}{\partial \theta}(x_n - [x_n]_0). \tag{5.1}$$

We can write this as $\Delta F = \nabla F \cdot \Delta\vec{x}$.

Further, consider a single step of gradient descent $\vec{x}_1 = \vec{x}_0 - \lambda\nabla F$ where $\lambda$ is the line search coefficient. We rewrite gradient descent in the form $\Delta\vec{x} = -\lambda\nabla F$.

Combining the tangent plane and gradient descent expressions yields

$$\Delta F = \nabla F \cdot \Delta\vec{x} = -\nabla F \cdot \lambda\nabla F = -\lambda\,\|\nabla F\|_2^2\,. \tag{5.2}$$

We solve for $\lambda$ to obtain our analytic line search coefficient.

This yields an iterative process. We perform gradient descent $\vec{x}_{n+1} = \vec{x}_n - \lambda_n \nabla F$ where we have analytic line search coefficient

$$\lambda_n = -\frac{F_t - F(\vec{x}_n)}{\|\nabla F\|_2^2} = \frac{F(\vec{x}_n) - F_t}{\|\nabla F\|_2^2}. \tag{5.3}$$

Now, we apply the above process to our test problem. Although we can use analytic gradients for the test problem, to mimic a problem in which analytic derivatives are not available, we use forward difference perturbations of one percent of the design ranges. We retain this gradient for the remainder of the line search process. We want to find $Y = 120$ N or some reasonably close value to this target; e.g. within one percent. At each step, we perform gradient descent

$$\begin{bmatrix} I_{i+1} \\ T_{i+1} \end{bmatrix} = \begin{bmatrix} I_i \\ T_i \end{bmatrix} - \lambda_i \begin{bmatrix} \frac{\partial Y}{\partial I} \\ \frac{\partial Y}{\partial T} \end{bmatrix} \tag{5.4}$$

with our updated step coefficient,

$$\lambda_i = \frac{Y_i - 120}{\frac{\partial Y}{\partial I}^2 + \frac{\partial Y}{\partial T}^2} = \frac{Y_i - 120}{\|\nabla Y\|_2^2} \tag{5.5}$$

where $T = \frac{\pi}{180}\theta$ and $T_i = \frac{\pi}{180}\theta_i$ are in radians. After three line search steps, we obtain the point $I = 0.1208$ mm and $\theta = 51.5553$ degrees where $Y = 120.9170$ N. We next obtain a tangent line approximation $L_1$ to the $Y = 120$ N contour of Figure 4.1(a).

**5.2. Moving in the Direction of Lower Variance.** Now that we have found where $Y \approx 120$ N, we follow the tangent line approximation $L_1$ in the direction of lower variance. We test three methods to direct movement.

**5.2.1. Linearized response function for calculating variance.** The First-Order, Second Moment (FOSM) variance formula [4] for the test problem is

$$\sigma^2_{\text{FOSM}} = \frac{\partial Y}{\partial K}^2 \sigma_K^2 + \frac{\partial Y}{\partial I}^2 \sigma_I^2 + \frac{\partial Y}{\partial \theta}^2 \sigma_\theta^2 + \frac{\partial Y}{\partial \mu}^2 \sigma_\mu^2. \tag{5.6}$$

We approximate the partial derivatives using forward difference calculations with one percent of the design range perturbations in $I$ and $\theta$. Correspondingly, we perturb $K$ by 1 N/mm and $\mu$ by 0.0035. Each FOSM calculation of the variance requires five function evaluations.

**5.2.2. Two Sample Ordinal Variance Estimate.** Figure 5.2 shows a crude approximation to the standard deviation contours using two samples at each point. The first sample is the quadruplet containing the nominal values at each point (i.e. $Y$ at $[\bar{K}, I_i, \theta_i, \bar{\mu}]$ where the subscript $i$ signifies the design space coordinates of point $i$ in Figure 5.1) and the second sample is the quadruplet where we perturb each variable by one standard deviation simultaneously (i.e. $Y$ at $[\bar{K} + \sigma_K, I_i + \sigma_I, \theta_i + \sigma_\theta, \bar{\mu} + \sigma_\mu]$). Each variance calculation requires only two function evaluations.

We compare the two sample ordinal standard deviation to the two hundred sample standard deviation. To make Figure 5.2, we perform $25 \times 25 \times 2 = 1,250$ function evaluations. We compare this to $25 \times 25 \times 200 = 125,000$ function evaluations used to create Figure 4.2(b). Although two sample ordinal is not able to predict the same absolute behavior of the contours, it is able to predict the relative behavior which is sufficient for the comparisons between successive points.

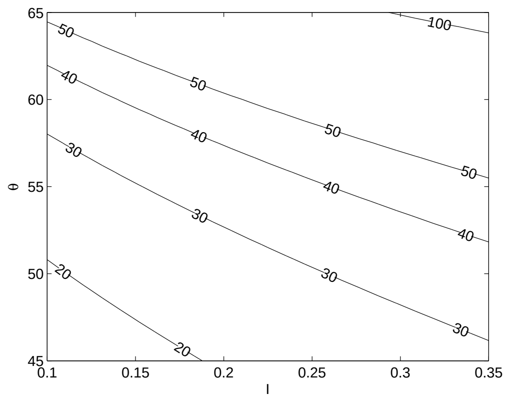


FIG. 5.2. *A standard deviation approximation using two previously chosen samples.*

**5.2.3. Sampling based on a Specified Confidence Level.** Using Latin hypercube sampling (LHS) that is implemented in DAKOTA [3], we perform an F-test. The null hypothesis is that the value of the variance at the third point is the same as the value of the variance at the second point. The alternative hypothesis is that the variance values at the two points are not equal.

We compare the variance at points a fixed distance apart on $L_1$. Based on the cost and benefit considerations, the following steps are one-fifth of the line length apart and tested at the 60 percent confidence level.

Starting from the second point, the first advance down the line takes us to a third point $I$ = 0.1315 mm and $\theta$ = 49.4045 degrees where $Y$ = 120.6423 N. We require 12 samples to determine that the two variance values are different with 60 percent confidence.

We take another step to the fourth point $I$ = 0.1422 mm and $\theta$ = 47.2537 degrees where $Y$ = 119.9849 N. Observe the proximity of $Y$ to 120 N. We do another F-test using LHS in DAKOTA. We find that 14 samples are required to confirm that the variance values at the third and fourth points are not equal with 60 percent confidence.

After another step, we obtain a fifth point $I$ = 0.1529 mm and $\theta$ = 45.1029 degrees. It takes 14 samples to confirm that the variance at the fourth point is not equal to the variance at the fifth point at the 60 percent confidence level. The value of $Y$ = 118.9819 N at the fifth point.

We can take one more truncated step down the line to the end point on the lower 45 degree edge of the design space where $I$ = 0.1534 mm. We call this point 6a. The friction force $Y$ = 118.9258 N is still within one percent of 120 N and we cannot move in any direction on the line to further decrease the variance.

**5.2.4. Summary of Procedure that satisfies the original 1 % Criterion.** Under the one percent criterion we set for moving down the line, the last step from the fifth point to point 6a completes our process. We find the final point by following one line $L_1$ without an additional side step. A summary of our results is included in Table 5.1. The table lists the point, its $Y$ value, and its standard deviation calculated by FOSM, two-sample ordinal sampling, and sampling with the sample size chosen based on sixty percent confidence in the F-test.

**5.3. Finding a new approximation to $Y$ = 120 N.** We set a new side step criterion of 1 N so that we can demonstrate a side step. We do not see holding to this tight of a criterion,

TABLE 5.1
*Iteration history of the algorithm with the original one percent criterion.*

| | $I$ (mm) | $\theta$ (degrees) | $Y$ (N) | FOSM | Ordinal | LHS |
|---|---|---|---|---|---|---|
| 1 | 0.225 | 55 | 261.1819 | 25.9131 | 36.6749 | |
| 2 | 0.1208 | 51.5553 | 120.9170 | 18.4662(5) | 22.3074(2) | 20.8599(12) |
| 3 | 0.1315 | 49.4045 | 120.6423 | 17.0504(5) | 20.7772(2) | 19.2423(12) |
| | | | | | | 18.0946(14) |
| 4 | 0.1422 | 47.2537 | 119.9849 | 15.8064(5) | 19.4389(2) | 16.8051(14) |
| 5 | 0.1529 | 45.1029 | 118.9819 | 14.7037(5) | 18.2567(2) | 15.6598(14) |
| 6a | 0.1534 | 45 | 118.9258 | 14.6540(5) | 18.2029(2) | |

but for the purpose of illustration we return to the fifth point where $Y$ is more than 1 N away from 120 N. Therefore, we find another approximation to $Y = 120$ N. We do this in one step of gradient descent with the analytic line search coefficient described in (5.5). This gives us a new point $I = 0.1541$ mm and $\theta = 45.1248$ degrees. At this sixth point, the value of $Y = 120.0013$ N. We obtain an updated tangent line approximation $L_2$ to $Y = 120$ N.

**5.4. Proximity to $\bar{Y} = 120$ N.** We continue in the direction of decreasing variance. We cannot complete a one-fifth of the line step without leaving the design space. Therefore we complete a truncated step to the lower edge of the design space. The seventh point we obtain is $I = 0.1548$ mm and $\theta = 45$ degrees where $Y = 119.9977$ N. Now we have found a point where the friction force is within 1 N of 120 N and we cannot move along the tangent line to further decrease variance. So, we are done with this part of the process.

Now, we must determine how close this point on the approximation to the $Y = 120$ N contour is to the $\bar{Y} = 120$ N contour. We want the true mean to be within two-fifths of the standard deviation of the target mean 120 N. We use two types of methods to study the mean at the final point. We begin with sampling for a specific confidence interval width. We also discuss point estimate methods.

**5.4.1. Sampled value of Mean used for Comparison.** To compare the accuracy of the methods we use to study the mean, we determine a reference mean by taking 100,000 LHS samples. We obtain a sample mean of $\bar{Y} = 120.1941$ N and a sample standard deviation of $s = 14.7477$ N. From these results, we obtain the 95 percent confidence interval on the mean (120.1027, 120.2855) N. Now we know that the true mean is within two-fifths of the sample standard deviation ($\frac{2}{5}s = 5.8991$ N) away from our target mean with 95 percent confidence.

**5.4.2. Sampling for specific confidence interval within $\frac{2}{5}\sigma$ of $\bar{Y}_{target}$.** We use a confidence interval on the mean to determine if the seventh point is near the $\bar{Y} = 120$ N contour. We attempt to fix the width of the confidence interval. We determine the number of samples required to obtain a 90 percent two-sided confidence interval on the mean with half-width $\frac{2}{5}s$ where $s$ is the sample standard deviation.

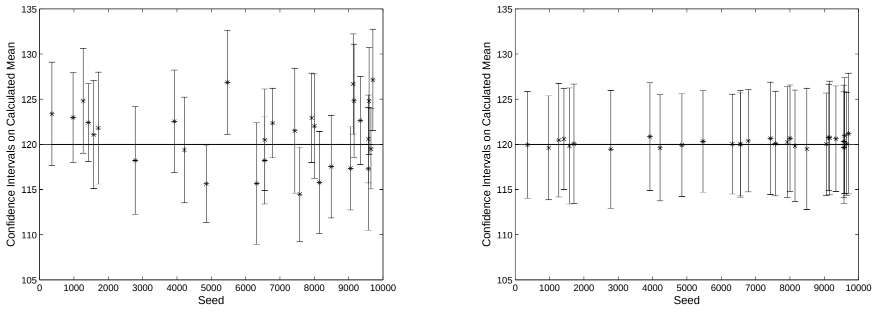
For the purpose of finding the required sample size $n$, we assume a standard normal distribution. Thus, we find $n$ such that $2/5 \approx \frac{t(1-\alpha/2;n-1)}{\sqrt{n}}$ where $t(1-\alpha/2; n-1)$ refers to Student's-t distribution. Using $\alpha = .1$, we find that we require $n = 19$ for the half-width to be approximately $\frac{2}{5}s$. Notice that the sample size obtained is not dependent on the test problem or the number of variables. If we wanted the half-width to be $\gamma s$, we find $n$ such that $\gamma \approx \frac{t(1-\alpha/2;n-1)}{\sqrt{n}}$.

We analyze the confidence intervals on the mean that are produced when $n = 19$ using simple random sampling (SRS) and LHS both implemented in DAKOTA. Among the sixty different samplings, we obtain five intervals which do not include 120 N. Figure 5.3 provides

TABLE 5.2
*Increasing accuracy of mean (first moment) estimates.*

| Method | Number of Function Evaluations | Mean $\bar{Y}$ (N) |
|---|---|---|
| Mean value | 1 | 119.9977 |
| Hong's "$2n$" | 8 | 120.1934 |
| Hong's "$2n + 1$" | 9 | 120.1933 |
| Two-point Rosenblueth"$2^n$" | 16 | 120.1935 |
| Three-point Seo-Kwak "$3^n$" | 81 | 120.1938 |
| LHS with 100,000 samples | 100,000 | 120.1941 |

a visual representation of the results from this sampling activity. The seed values are found using the MATLAB implementation of the Mersenne Twister pseudorandom number generator. Initially, the function is reset using the seed value 5489. We take the numbers produced by the routine and multiply them by 10,000. We obtain the current seed by taking the ceiling function of that product.



(a) Two-sided 90 percent confidence intervals using SRS.

(b) Two-sided 90 percent confidence intervals using LHS.

FIG. 5.3. *Examination of the two-sided 90 percent confidence intervals of the mean.*

Among our thirty SRS trials, the SRS estimates lead us to find that more than 10 percent of the intervals did not contain the true mean. Previous experience with a larger number of trials [9] indicates that you cannot expect $100(1 - \alpha)$ percent SRS confidence intervals to contain the true mean the advertised $100(1 - \alpha)$ percent of the time. We strongly recommend using LHS over SRS. Based on the results, we believe that the true mean is within $\frac{2}{5}\sigma$ of the target mean 120 N.

**5.4.3. Statistical-Moment Generation Approach (Response Mean by Optimal Placement and Weighting).** We use point estimate methods (PEMs) to determine the mean (first moment) by optimal placement and weighting of samples. These methods are also referred to as statistical-moment generating methods. Table 5.2 provides an overview of the results. Observe the proximity of these results to the 100,000 sample mean. We observe that these methods provide fairly high accuracy at low cost. Another benefit of using PEMs is that the same samples used to compute the mean can also be used to compute the variance and standard deviation so that no further function evaluations are necessary.

The mean value estimate [1] is 119.9977 N. The final point is the only sample.

Now, we use a $2n$ PEM [6] where $n$ is the number of variables. For our example, $n = 4$.

This case corresponds to $m = 2$. Let $k = 1, \ldots, n$ and $i = 1, \ldots, m$. We sample the points

$$x_{k,i} = \bar{x}_k + \xi_{k,i} s_k \tag{5.7}$$

where $\bar{x}_k$ is the nominal value of the $k^{th}$ variable and $s_k$ is the standard deviation of the $k^{th}$ variable. Under the assumption of normality, we have

$$\xi_{k,i} = (-1)^{3-i} \sqrt{n} \tag{5.8}$$

and equal weighting is used so that the weights of each point are defined by $p_{k,i} = \frac{1}{2n}$. This method yields the mean 120.1934 N.

A $2n + 1$ method [6] uses the center point as well. This is the $m = 3$ case. To clarify the formula, the kurtosis of a normal distribution is considered to be three. We use the same point-generation formula (5.7) as before, but now, under the assumption of normality, we have

$$\begin{cases} \xi_{k,i} = (-1)^{3-i} \sqrt{3}, \; i = 1, 2 \\ \xi_{k,3} = 0. \end{cases} \tag{5.9}$$

The weights also change. Now, $p_{k,1} = p_{k,2} = \frac{1}{6}$ and the middle point's weight is $4 \times -\frac{1}{12} = -\frac{1}{3}$. From this exercise, we obtain the mean estimate 120.1933 N.

We try a $2^n$ [10] method. Let $i = 1, 2$ and $k = 1, \ldots, n$. We use (5.7) just as before, but now

$$\xi_{k,i} = (-1)^{3-i}. \tag{5.10}$$

We have equal weighting so $p_{k,i} = \frac{1}{2^n}$. We find the mean estimate 120.1935 N.

The final estimation method we try is a three-point Seo-Kwak [11]. Now $i = 1, 2, 3$ and $k = 1, \ldots, n$. We use (5.7) with

$$\begin{cases} \xi_{k,i} = (-1)^{3-i} \sqrt{3}, \; i = 1, 2 \\ \xi_{k,3} = 0. \end{cases} \tag{5.11}$$

The Gauss-Hermite three-point weights are used. Therefore $p_{k,1} = p_{k,2} = \frac{1}{6}$ and $p_{k,3} = \frac{2}{3}$. A product weighting system is used. The first-moment result from this method is 120.1938 N.

**5.5. Iteration History.** Table 5.3 lists the iteration history starting from the middle of the design space. The variance and standard deviation provided are found using FOSM. The partial derivatives are computed by a forward difference approximation. For the design variables $I$ and $\theta$, the perturbations are one percent of their design ranges. For $K$, the perturbation is 1 N/mm. For the friction coefficient $\mu$, the perturbation is 0.0035. For a visual history of the algorithm, refer back to Figure 5.1.

**5.6. Cost.** We first determine the cost of moving along the $Y = 120$ N contours. We then discuss the variance comparison cost along the way. We end with the cost of the studies on the mean. Observe that this is the cost associated with the 1.2 N side step criterion.

Performing gradient descent with an analytic line search to get our first point on $Y = 120$ N costs six function evaluations. We use three more to determine $L_1$. We follow this line to the fifth point while we check to make sure we are within some tolerance of 120 N. This costs four function evaluations. Altogether we spend 13 function evaluations to move down $L_1$.

Now, we consider the cost of the variance comparisons. We have already computed the function value at all five points on the tangent line so we use the previously computed friction force values in the variance estimates. Using linear approximations costs us $5 \times 4 = 20$

TABLE 5.3
*Iteration history the algorithm.*

| Iteration | $I$ (mm) | $\theta$ (degrees) | $Y$ (N) | $s^2$ ($N^2$) | $s$ (N) |
|---|---|---|---|---|---|
| 1 | 0.225 | 55 | 261.1819 | 671.4899 | 25.9131 |
| 2 | 0.1208 | 51.5553 | 120.9170 | 341.0013 | 18.4662 |
| 3 | 0.1315 | 49.4045 | 120.6423 | 290.7151 | 17.0504 |
| 4 | 0.1422 | 47.2537 | 119.9849 | 249.8437 | 15.8064 |
| 5 | 0.1529 | 45.1029 | 118.9819 | 216.1979 | 14.7037 |
| 6a | 0.1534 | 45 | 118.9258 | 214.7400 | 14.6540 |
| 6 | 0.1541 | 45.1248 | 120.0013 | 217.2102 | 14.7380 |
| 7 | 0.1548 | 45 | 119.9977 | 215.4828 | 14.6793 |

more function evaluations. Using the ordinal sampling requires $5 \times 1 = 5$ additional function evaluations. Alternatively, we do the F-tests to compare the points that are one-fifth of the line apart using 66 function evaluations. The values used in the F-tests were generated by LHS which is not incremental. If we had used SRS, we could take advantage of incremental sampling and the cost would be 54. However, we do not know if the number of samples required to deem that the variances differed would be the same for SRS.

We tally the number of function evaluations needed to perform the analysis of the mean at the final point. We attempt to fix the confidence interval magnitude. To do this and do a check for false positives/negatives, we require $19 \times 2 = 38$ function evaluations. However, one confidence interval computed with 19 LHS results may be sufficient as it was in the example we provided. We save significantly by using the PEMs. Since we already computed the friction force $Y$ at the final point, mean value is free. For the same reason, using $2n + 1$ requires only 8 more function evaluations. Since $2n$ and $2n + 1$ are not sampled at the same points, we use 8 more to get the $2n$ result. Sixteen samples are averaged to obtain the two-point Rosenblueth result. Since the $3^n$ uses the center sample and the same samples at the center of each face used for $2n + 1$, we use only 72 more function evaluations to get the Seo-Kwak estimate.

**6. Comparison of results to other methods.** We compare the results of a few other methods to the result of the algorithm when a 1.2 N side step criterion is used. We use 1,000 LHS samples to determine the mean and standard deviation at the returned point for each method. The sample mean at the final point is 119.1069 N and the sample standard deviation is 14.7263 N.

Using DAKOTA we perform an exhaustive search to find the point of best variance where the mean value of the friction force is approximately 120 N. We implement the nested optimization routines in DAKOTA. The outer loop is an optimization algorithm while the mean and variance used in the constraints and objective function are determined in the inner loop.

The outer loop is a Coliny division of rectangles (DIRECT) optimization routine. The global search balancing parameter is set to zero while the local search balancing parameter is set to $1 \times 10^{-8}$. We test DIRECT using two inner loops.

First, the mean and variance are determined by 10 LHS samples in the inner loop. The objective of DIRECT is to minimize variance subject to the constraint that the mean be within 1 N of the target mean. We set a solution threshold of 15. This nested routine performs 1,250 function evaluation and returns the point $I = 0.1519$ mm and $\theta = 45.3705$ degrees. These results correspond to a mean of 119.6255 N and a standard deviation of 14.9158 N.

Next, we use the DAKOTA reliability package [1] to further confirm our results. We aim to minimize $E[| \ Y - 120 \ |] + C\sigma$ with a solution threshold of 55. The mean and standard deviation are computed using mean value with a gradient stepsize of $1 \times 10^{-4}$. We set the convergence tolerance to $1 \times 10^{-4}$ and the threshold delta to $1 \times 10^{-8}$. This optimization is sensitive to the formulation of the objective function. The $C$ reflects the $C\sigma$ reliability [2]. When we test $C = 3$, we find the point $I = 0.1365$ mm and $\theta = 48.3338$ degrees using 105 function evaluations. Here the mean is 120.3002 N and the standard deviation is 16.5004 N.

We test another optimization routine that is implemented by the Coliny pattern search in DAKOTA. The objective is to minimize variance subject to the constraint that the mean be within 1 N of the target mean 120 N. We set a solution threshold of 15. The solution accuracy is chosen to be $1 \times 10^{-8}$. We pick an initial delta of .5. The threshold delta is set to $1 \times 10^{-13}$. We choose a contraction factor of .85. Using 1,360 function evaluations, the routine yields the point $I = 0.1535$ mm and $\theta = 45.4438$ degrees where the mean is 121.1675 N and the standard deviation is 14.9873 N.

Table 7.1
*Costs of the optimization.*

| Method | Task | Options | Cost |
|---|---|---|---|
| Algorithm | | | |
| | Find $Y = Y_{\text{target}}$ | | 13 |
| | Lower Variance | | |
| | | FOSM | 20 |
| | | Ordinal | 5 |
| | | Hypothesis Testing | 66 |
| | Determine Mean | | |
| | | Fixed-width CI | 19 |
| | | Mean value | 0 |
| | | Hong's "2n" | 8 |
| | | Hong's "2n+1" | 8 |
| | | Two-point Rosenblueth "$2^n$" | 16 |
| | | Three-point Seo-Kwak "$3^n$" | 72 |
| DIRECT | | using LHS | 1,250 |
| | | with reliability | 113 |
| Pattern search | | | 1,360 |

**7. Cost Comparison.** The cost is the number of function evaluations required. In other words, we compare how many times we need to compute the friction force $Y$ for each method. It is clear that the cost of our algorithm is dependent on which methods are used for each task and the problem. We report the cost of performing the algorithm when the original 1.2 N side step criterion is used.

We see that all of the methods yield similar results. The main reason to use the algorithm we have provided is the ultimate cost savings.

The cost of the algorithm is dependent on which methods are used. The entire algorithm can be implemented in 26 function evaluations using two sample ordinal variance comparisons and Hong's "$2n$" estimate of the mean and standard deviation at the final point. The maximum cost of the algorithm is 151 function evaluations when F-test variance comparisons are used and the final mean and standard deviation estimate are determined by "$3^n$" Seo-Kwak. For the types of applications we consider, the three-point Seo-Kwak is not cost-performance competitive to the two-point Rosenblueth. We suggest using two sample ordinal variance comparisons and the "$2^n$" two-point Rosenblueth method to estimate the mean and standard deviation at the final point. The number of samples required for the PEMs is dependent on the number of variables $n$. If $n \geq 5$, we suggest using the fixed width confidence interval.

Overall, we see considerable savings over the stand alone methods discussed in the previous two sections. Table 7.1 displays the costs of the various methods.

**8. Conclusions.** We have described an algorithm designed to reduce the number of function evaluations required to perform optimization under uncertainty. This method minimizes variance while ensuring that a prescribed mean target level of system output is achieved. We applied the optimization procedure to an automotive device design robustness problem. We compared the results and cost of our algorithm to other algorithms. We found that the suggested algorithm finds similar or better results at a lower cost.

## REFERENCES

[1] M. Eldred, H. Agarwal, V. Perez, S. Wojtkiewicz Jr., and J. Renaud, *Investigation of reliability method formulations in DAKOTA/UQ*, in Proceedings of the 9th ASCE Joint Specialty Conference on Probabilistic Mechanics and Structural Reliability, Albuquerque, NM, July 26–28 2004.

[2] M. Eldred, A. Giunta, S. Wojtkiewicz Jr., and T. Truncano, *Formulations for surrogate-based optimization under uncertainty*, in Proceedings of the 9th AIAA/ISSMO Symposium on Multidisciplinary Analysis and Optimization, no. AIAA-2002-5585, Atlanta, GA, Sept. 4–6 2002.

[3] M. S. Eldred, B. M. Adams, D. M. Gay, L. P. Swiler, K. Haskell, W. J. Bohnhoff, J. P. Eddy, W. E. Hart, J.-P. Watson, J. D. Griffin, P. D. Hough, T. G. Kolda, P. J.Williams, and M. L. Martinez-Canales, *DAKOTA, a multilevel parallel object-oriented framework for design optimization, parameter estimation, uncertainty quantification, and sensitivity analysis version 4.1 users manual*, Tech. Report SAND2006-6337, Sandia National Laboratories, October 2007. Updated September 2007.

[4] A. Haldar and S. Mahadevan, *Probability, Reliability, and Statistical Methods in Engineering Design*, John Wiley and Sons, New York, 2000.

[5] M. E. Harr, *Probabilistic estimates for multivariate analyses*, Applied Mathematical Modeling, 13 (1989), pp. 313–318.

[6] H. Hong, *An efficient point estimate method for probabilistic analysis*, Reliability Engineering and System Safety, 59 (1998), pp. 261–267.

[7] Machine Design.com, *Fundamentals of annular snap-fit joints*. `http://machinedesign.com/ContentItem/61167/FundamentalsofAnnularSnapFitJoints.aspx`, January 2005. Penton Media, Inc.

[8] J. Neter, W. Wasserman, and M. Kutner, *Applied Linear Statistical Models*, Irwin, Homewood, IL, 1985.

[9] V. J. Romero and C.-H. Chen, *Refinements in a new adaptive ordinal approach to continuous-variable probabilistic optimization*, AIAA Journal, (2007). To appear.

[10] E. Rosenblueth, *Two-point estimates in probability*, Applied Mathematical Modeling, 5 (1981), pp. 329–335.

[11] H. S. Seo and B. M. Kwak, *Efficient statistical tolerance analysis for general distributions using three-point information*, Intl. Journal of Production Research, 40 (4), pp. 931–944.

[12] S.-C. Tsai, *Taguchi S/N ratios and direct robustness measurement for computational robust design*, in SAE2006-01-0738, Detroit, MI, April 3–6 2006, 2006 SAE World Congress.

[13] L. Wang, D. Beeson, and G. Wiggs, *Efficient moment and probability distribution estimation using the point estimate method for high-dimensional engineering problems*, in 48th AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference, Honolulu, Hawaii, April 23–26 2007.

# CALIBRATION AND UNCERTAINTY ANALYSIS FOR EXPENSIVE COMPUTER SIMULATIONS

JOHN M. MCFARLAND*, LAURA P. SWILER†, AND VICENTE J. ROMERO‡

**Abstract.** The use of complex simulation models, including finite element and other PDE solvers, is continuing to increase in prevalence within the scientific community. These simulations are often used for prediction, parameter studies, and high-consequence decision making, and may be characterized by a large number of input parameters, long run times, and high-dimensional output. In many cases, experiments may be conducted under a set of "moderate" conditions, and such results may be compared against the corresponding model predictions. When these experimental data are used to improve the predictive capability of the model (perhaps by making inference about internal model parameters), we call this model calibration. This work explores model calibration for realistic engineering simulations, with an emphasis on taking a comprehensive and understandable account of various uncertainties present using the methods of Bayesian analysis. We employ the use of Gaussian process models as surrogates for the expensive simulation. In addition, we illustrate a method by which prescribed uncertainties at the modeling level can be accounted for in the calibration under uncertainty process. Our complete methodology is thoroughly illustrated for a thermal simulation of a complex foam model, which includes a large database of experimentally observed response values over time and space.

## 1. Introduction.

The importance of uncertainty in the modeling and simulation process is often overlooked. No model is a perfect representation of reality, so it is important to ask how imperfect a model is before it is applied for prediction. The scientific community relies heavily on modeling and simulation tools for forecasting, parameter studies, design, and decision making. However, these are all activities which can strongly benefit from meaningful representations of modeling uncertainty. For example, forecasts can contain error bars, designs can be made more robust, and decision makers can be better-informed when modeling uncertainty is quantified to support these activities.

The set of activities which involve the quantification of uncertainty in the modeling and simulation process includes verification, validation, calibration, and uncertainty propagation. Verification involves the comparison of a computational implementation with a conceptual model, in order to "verify" the implementation and assess the amount of error introduced via numerical processes. Validation, on the other hand, is a process for comparing the computational implementation of a model against experimentally observed outcomes: this is another opportunity to quantify errors and uncertainties. Similarly, calibration involves comparing the implementation of a model with observations, but the objective is to use this comparison to make inferences about unknown parameters which govern the computational implementation. Uncertainty propagation is simply the process of determining the uncertainty on the model output that is implied by uncertainty on the model inputs.

The purpose of this paper is illustrate a methodology wherein the Bayesian framework can be used effectively for the calibration of complex computer simulations having long run times and highly multivariate output. The Bayesian framework is attractive because it provides a comprehensive, quantitative treatment of uncertainty: the solution is not just a single parameter set that best fits the observed experimental data, but a probability distribution that represents the amount of uncertainty present in the solution. For example, large modeling sensitivities or a large amount of experimental observations may result in a solution with very little residual uncertainty, whereas in other cases there may be a wide range of parameters that yield comparable fits to the data.

*Vanderbilt University, john.m.mcfarland@vanderbilt.edu
†Sandia National Laboratories, lpswile.sandia.gov
‡Sandia National Laboratories, vjromer@sandia.gov

## 2. Bayesian model calibration.

Model calibration is a particular type of inverse problem in which we are interested in finding values for a set of computer model inputs which result in computer model outputs that agree well with observed data. There are several ways to approach the model calibration problem, and one of the most straightforward is to formulate it as a non-linear least squares optimization problem, in which we want to minimize the sum of the squares of the residuals between the model predictions and the observed data. This approach can be attractive because of its simplicity, but it also has several drawbacks:

1. Finding the set of model inputs which minimizes the sum of squares may require a large number of evaluations of the model (depending on the type of optimization algorithm being employed). When the model is very expensive to run, this approach may not even be feasible.
2. There may be a wide range of model inputs which provide comparable fits to the observed data (this is sometimes termed the problem of uniqueness).
3. Small changes in some of the model inputs may cause drastic variations of the model output, resulting in an ill-posed optimization problem.

Further, approaching the calibration problem as a least-squares optimization problem will yield only one solution, and it can be difficult to construct meaningful information about the uncertainty associated with this solution (although some approaches have been attempted, as in [14]). Thus, there would be a large amount of utility in any method which overcomes the difficulties associated with the non-linear least squares approach, and provides a more comprehensive treatment of the uncertainties present. Fortunately, the field of Bayesian analysis provides such a method.

The fundamental concept of Bayesian analysis is that unknown variables are treated as random variables. The power of this approach is that the established mathematical methods of probability theory can then be applied. Uncertain variables are given “prior” probability distribution functions, and these distribution functions are refined based on the available data, so that the resulting “posterior” distributions represent the new state of knowledge, in light of the observed data. While the Bayesian approach can be computationally intensive in many situations, it is attractive because it provides a very comprehensive treatment of uncertainty. More details behind the Bayesian approach are given in Section 2.1.

### 2.1. Introduction to Bayesian Analysis.

Bayesian statistical analysis differs from classical (or frequentist) statistics fundamentally by the two camps’ interpretations of probability. In classical statistics, the meaning of probability is directly related to frequency of occurrence. What sets Bayesians apart is that they allow probability and probability distributions to connote belief or uncertainty about uncertain parameters. Thus, Bayesian analysis begins with what is known as a “prior” distribution for the uncertain parameter, denoted $\pi(\theta)$. Knowledge about the uncertain parameter is then updated by observations, $D$ to arrive at what is called the “posterior” distribution of $\theta$. This process is expressed formally through what is known as Bayes’ theorem:

$$f(\theta \mid D) = \frac{\pi(\theta) f(D \mid \theta)}{\int \pi(\theta) f(D \mid \theta) d\theta}, \tag{2.1}$$

where $f(D \mid \theta)$ is known as the likelihood function of $\theta$, and is commonly denoted $L(\theta)$ because the data in $D$ hold a fixed value once observed.

The meaning of Bayes’ theorem is that the posterior distribution of $\theta$ is proportional to the prior times the likelihood (note that the integral in the denominator functions to normalize the posterior distribution so that is has a total area of 1). It is worth noting that while many classical statisticians argue fervently against the use of Bayesian analysis because of the apparent subjectivity present in formulating prior distributions, there are many cases in which well-known classical results can be derived using Bayesian analysis. This is often the case when Bayesian analysis is applied using what are known as vague, non-informative, or reference prior distributions (which are most commonly given by uniform or log-uniform probability distributions), whose purpose is to represent an absence of prior knowledge, so that the posterior distribution is a function of the data only.

The primary computational difficulty in applying Bayesian analysis is the evaluation of the integral in the denominator of Eq. (2.1), particularly when dealing with multiple variables. When closed form solutions are not available, computational sampling techniques such as Markov Chain Monte Carlo (MCMC) sampling are often used [11, 4, 7, 9].

**2.2. Bayesian analysis for model calibration.** Consider that we are interested in making inference about a set of computer model inputs $\boldsymbol{\theta}$. Now let us represent our simulation by the forward model operator $G(\boldsymbol{\theta}, \boldsymbol{s})$, where the vector of inputs $\boldsymbol{s}$ represents a set of "scenario-descriptor" inputs, which may typically represent boundary conditions, initial conditions, geometry, etc. Reference [10] terms these inputs "variable inputs", because they take on different values for different realizations of the system. Thus, $y = G(\boldsymbol{\theta}, \boldsymbol{s})$ is the response quantity of interest associated with our model. Also, we assume that the value of the calibration inputs $\boldsymbol{\theta}$ should not depend on $\boldsymbol{s}$, the particular realization of the system being modeled (this point is discussed in more detail by [10]).

Now consider a set of $n$ experimental measurements

$$\boldsymbol{d} = d_1, \ldots, d_n,$$

which are to be used to calibrate the simulation. Note that each experimental measurement corresponds to a particular value of the scenario-descriptor inputs, $\boldsymbol{s}$, and we assume that these values are known for each experiment. Thus, we are interested in finding those values of $\boldsymbol{\theta}$ for which the simulation outputs $(G(\boldsymbol{\theta}, \boldsymbol{s}_1), \ldots, G(\boldsymbol{\theta}, \boldsymbol{s}_n))$ agree well with the observed data in $\boldsymbol{d}$. But as mentioned above, we are interested in more than simply a point estimate for $\boldsymbol{\theta}$: we would like a comprehensive assessment of the uncertainty associated with this estimate.

First, we define a probabilistic relationship between the model output, $G(\boldsymbol{\theta}, \boldsymbol{s})$, and the observed data, $\boldsymbol{d}$:

$$d_i = G(\boldsymbol{\theta}, \boldsymbol{s}_i) + \varepsilon_i, \tag{2.2}$$

where $\varepsilon_i$ is a random variable that can encompass both measurement errors on $d_i$ and modeling errors associated with the simulation $G(\boldsymbol{\theta}, \boldsymbol{s})$. The most frequently used assumption for the $\varepsilon_i$ is that they are i.i.d $N(0, \sigma^2)$, which means that the $\varepsilon_i$ are independent, zero-mean Gaussian random variables, with variance $\sigma^2$. Of course, more complex models may be applied, for instance enforcing a parametric dependence structure among the errors.

The probabilistic model defined by Eq. (2.2) results in a likelihood function for $\boldsymbol{\theta}$ which is the product of $n$ normal probability density functions:

$$L(\boldsymbol{\theta}) = f(\boldsymbol{d} \mid \boldsymbol{\theta}) = \prod_{i=1}^{n} \frac{1}{\sigma\sqrt{2\pi}} \exp\left[-\frac{(d_i - G(\boldsymbol{\theta}, \boldsymbol{s}_i))^2}{2\sigma^2}\right]. \tag{2.3}$$

We can now apply Bayes' theorem (Eq. (2.1)) using the likelihood function of Eq. (2.3) along with a prior distribution for $\boldsymbol{\theta}$, $\pi(\boldsymbol{\theta})$, to come up with a posterior distribution, $f(\boldsymbol{\theta} \mid \boldsymbol{d})$, which represents our belief about $\boldsymbol{\theta}$ in light of the data $\boldsymbol{d}$:

$$f(\boldsymbol{\theta} \mid \boldsymbol{d}) \propto \pi(\boldsymbol{\theta}) L(\boldsymbol{\theta}). \tag{2.4}$$

The posterior distribution for $\boldsymbol{\theta}$ represents our complete state of knowledge, and may even include effects such as multiple modes, which would represent multiple competing hypotheses about the true (best-fitting) value of $\boldsymbol{\theta}$. Summary information can be extracted from the posterior, including the mean (which is typically taken to be the the "best guess" point estimate) and standard deviation (a representation of the amount of residual uncertainty). We can also extract one or two-dimensional marginal distributions, which simplify visualization of the features of the posterior.

However, as discussed in Section 2.1, the posterior distribution can not usually be constructed analytically, and this will almost certainly not be possible when a complex simulation model appears inside the likelihood function. One of the more popular numerical techniques for constructing the posterior distribution is Markov Chain Monte Carlo (MCMC) simulation [11, 4, 7, 9]. Unfortunately, though, MCMC simulation requires hundreds of thousands of evaluations of the likelihood function, which in the case of model calibration equates to hundreds of thousands of evaluations of the computer model $G(\cdot, \cdot)$. For most realistic models, this number of evaluations will not be feasible. In such situations, the analyst must usually resort to the use of a more inexpensive surrogate (a.k.a response surface approximation) model. Such a surrogate might involve reduced order modeling (e.g., a coarser mesh) or data-fit techniques such as Gaussian process (a.k.a kriging) modeling.

This work adopts the approach of using a Gaussian process surrogate to the true simulation. We find such an approach to be an attractive choice for use within the Bayesian calibration framework for several reasons:

1. The Gaussian process model is incredibly flexible, and can be used to fit data associated with virtually any functional form.
2. The Gaussian process model is stochastic, thus providing both an estimated response value and an uncertainty associated with that estimate. Conveniently, the Bayesian framework allows us to take account of this uncertainty.
3. With regards to fit accuracy, the Gaussian process model has been shown to be competitive with most other modern data fit methods, including Bayesian neural networks and Multiple Adaptive Regression Splines [12, 8], and it can represent functions with multiple inputs.

For model calibration with an expensive simulation, the uncertainty associated with the use of a Gaussian process surrogate can be accounted for through the likelihood function. Through the assumptions used for Gaussian process modeling, the response conditional on a set of observed "training points" follows a multivariate normal distribution. For a discrete set of new inputs, this response is characterized by a mean vector and a covariance matrix (see [12]). Let us denote the mean vector and covariance matrix corresponding to the inputs $(\boldsymbol{\theta}, \boldsymbol{s}_1), \ldots, (\boldsymbol{\theta}, \boldsymbol{s}_n)$ as $\boldsymbol{\mu}_{GP}$ and $\boldsymbol{\Sigma}_{GP}$, respectively. It is easy to show that the likelihood function for $\boldsymbol{\theta}$ is then given by a multivariate normal probability density function (note that the likelihood function of Eq. (2.3) can also be expressed as a multivariate normal probability density, with $\boldsymbol{\Sigma}$ diagonal):

$$L(\boldsymbol{\theta}) = (2\pi)^{-n/2} |\boldsymbol{\Sigma}|^{-1/2} \exp\left[-\frac{1}{2}(\boldsymbol{d} - \boldsymbol{\mu}_{GP})^T \boldsymbol{\Sigma}^{-1} (\boldsymbol{d} - \boldsymbol{\mu}_{GP})\right], \tag{2.5}$$

where $\boldsymbol{\Sigma} = \sigma^2 \boldsymbol{I} + \boldsymbol{\Sigma}_{GP}$, so that both $\boldsymbol{\mu}_{GP}$ and $\boldsymbol{\Sigma}$ depend on $\boldsymbol{\theta}$.

Simply put, since the uncertainty associated with the surrogate model is independent of the modeling and observation uncertainty captured by the $\varepsilon_i$, the covariance of the Gaussian process predictions ($\boldsymbol{\Sigma}_{GP}$) simply adds to the covariance of the error terms ($\sigma^2 \boldsymbol{I}$). As mentioned before, if a more complicated error model is desired (i.e. one that does not assume the errors to be independent of each other), we can replace $\sigma^2 \boldsymbol{I}$ by a full covariance matrix.

Also, in some cases where there is a very large amount of experimental data available, we may even want to model different "segments" of the output (e.g., different spatial locations or different time intervals) using separate, independent Gaussian process surrogates. In such a case, the likelihood function is a product of multivariate normal densities, where each density contains a particular partition of $\boldsymbol{d}$ and the corresponding surrogate predictions $\boldsymbol{\mu}_{GP}$ and $\boldsymbol{\Sigma}_{GP}$. Such a formulation may improve the accuracy of and decrease the uncertainty in the surrogates because they are more localized, but the implementation is somewhat more cumbersome.

**2.2.1. Prescribed input uncertainties.** In some cases it may be of interest to study how the results of a calibration analysis are affected by additional modeling uncertainties. In most cases we would do so in the Bayesian setting by augmenting the set of calibration parameters $\boldsymbol{\theta}$ with the additional uncertain model inputs. If the data $\boldsymbol{d}$ do not provide any information about these additional uncertain inputs, then they will essentially be sampled over their prior distribution, potentially resulting in an increase in the uncertainty in the original calibration parameters. On the other hand, if the data $\boldsymbol{d}$ do provide information about the additional inputs, then their posterior distribution will most likely reflect less uncertainty than their prior. However, if we are strictly interested in the effect of additional prescribed input uncertainties, such inputs can not be treated as calibration inputs, because their posterior may not match the prescribed distribution of interest. Thus, this section presents a method which allows the Bayesian calibration analysis to take account of prescribed uncertainties for additional model inputs.

Let us denote those inputs to the simulation $G(\cdot)$ having prescribed probability distributions by $\boldsymbol{\xi}$. Thus, our simulation model is now a function of the calibration inputs, the scenario-descriptor inputs, and the inputs with prescribed distributions: $y = G(\boldsymbol{\theta}, s, \boldsymbol{\xi})$. Denote the probability density function associated with $\boldsymbol{\xi}$ by $f(\boldsymbol{\xi})$. In order to develop the posterior distribution for $(\boldsymbol{\theta}, \boldsymbol{\xi})$ in which the distribution of $\boldsymbol{\xi}$ is not refined by $\boldsymbol{d}$, we must assume artificially that the data $\boldsymbol{d}$ are statistically independent of $\boldsymbol{\xi}$. Whether or not this is true in reality can be checked by treating $\boldsymbol{\xi}$ as a calibration parameter in $\boldsymbol{\theta}$, but by artificially enforcing the assumption, the parameters $\boldsymbol{\xi}$ are held to the prescribed distribution $f(\boldsymbol{\xi})$.

By assuming that $\boldsymbol{\xi}$ is independent of $\boldsymbol{d}$, we have:

$$f(\boldsymbol{\theta}, \boldsymbol{\xi} \mid \boldsymbol{d}) \propto \pi(\boldsymbol{\theta}) L(\boldsymbol{\theta}) f(\boldsymbol{\xi}).$$

Since the simulation output is a function of $\boldsymbol{\xi}$, $L(\boldsymbol{\theta})$ is as well, so for clarity we write $L(\boldsymbol{\theta}; \boldsymbol{\xi})$[1], which yields:

$$f(\boldsymbol{\theta}, \boldsymbol{\xi} \mid \boldsymbol{d}) \propto \pi(\boldsymbol{\theta}) L(\boldsymbol{\theta}; \boldsymbol{\xi}) f(\boldsymbol{\xi}). \tag{2.6}$$

Ultimately, though, we are interested in the posterior of $\boldsymbol{\theta}$ after marginalizing over the "nuisance" variable $\boldsymbol{\xi}$, so we want

$$f(\boldsymbol{\theta} \mid \boldsymbol{d}) \propto \int \pi(\boldsymbol{\theta}) L(\boldsymbol{\theta}; \boldsymbol{\xi}) f(\boldsymbol{\xi})\, d\boldsymbol{\xi}. \tag{2.7}$$

This marginalization is trivial if $f(\boldsymbol{\theta}, \boldsymbol{\xi} \mid \boldsymbol{d})$ is constructed using Markov Chain Monte Carlo sampling. One possibility for constructing $f(\boldsymbol{\theta}, \boldsymbol{\xi} \mid \boldsymbol{d})$ is to use a component-wise scheme to sequentially sample each component of $(\boldsymbol{\theta}, \boldsymbol{\xi})$ from its respective full conditional

[1]Although it is tempting to write $L(\boldsymbol{\theta}, \boldsymbol{\xi})$, we avoid doing so because this is really $f(\boldsymbol{d} \mid \boldsymbol{\theta}, \boldsymbol{\xi})$; since $\boldsymbol{\xi}$ is (assumed to be) statistically independent of $\boldsymbol{d}$, this would reduce to $f(\boldsymbol{d} \mid \boldsymbol{\theta}) = L(\boldsymbol{\theta})$. Thus, we write $L(\boldsymbol{\theta}; \boldsymbol{\xi})$ to emphasize that it is a function of $\boldsymbol{\xi}$, but there is no statistical relationship between $\boldsymbol{\xi}$ and $\boldsymbol{d}$.

distribution. Each component of $\boldsymbol{\theta}$ can be sampled using the Metropolis algorithm, by sampling the $i$th component from its full conditional:

$$f(\theta_i \mid \boldsymbol{\theta}_{-i}, \boldsymbol{\xi}, \boldsymbol{d}) \propto \pi(\boldsymbol{\theta}) L(\boldsymbol{\theta}; \boldsymbol{\xi}), \tag{2.8}$$

where $\boldsymbol{\theta}_{-i}$ contains all components of $\boldsymbol{\theta}$ except for $\theta_i$. Notice that $f(\boldsymbol{\xi})$ does not appear in Eq. (2.8) because it does not depend on $\boldsymbol{\theta}$.

Further, if the joint distribution of $\boldsymbol{\xi}$ is sampleable (in particular, if the components of $\boldsymbol{\xi}$ are independent, with sampleable marginals), the vector $\boldsymbol{\xi}$ can be directly sampled at each iteration. This is because the full conditional of $\boldsymbol{\xi}$ is equal to $f(\boldsymbol{\xi})$, so at each iteration we draw a sample of $\boldsymbol{\xi}$ from $f(\boldsymbol{\xi})$, which is its full conditional.

In short, the process for accounting for prescribed input uncertainties within the Bayesian calibration analysis is very simple, given that Markov Chain Monte Carlo is used to construct the posterior for $\boldsymbol{\theta}$. To account for the additional total uncertainty introduced by the inputs, $\boldsymbol{\xi}$, having prescribed uncertainties, we simply sample a random realization of $\boldsymbol{\xi}$ at each iteration of the MCMC sampler.

**3. Case study: Calore thermal simulation.** To illustrate the Bayesian calibration methodology, we use a thermal simulation of a canister containing a mock weapons component encapsulated by a foam insulation (a.k.a. "foam in a can"). A series of experiments have been conducted at Sandia National Laboratories in an effort to support the physical characterization and modeling of thermally decomposing foam [6]. Several illustrations of this setup are shown in Figure 3.1. An associated thermal model, using the Calore code, is described in [13]. Calore is a computational heat transfer code being developed at Sandia under the ASC (Advanced Simulation and Computing Program) of the NNSA (National Nuclear Security Administration) [3]. Calore approximates linear and nonlinear continuum models of heat transfer, with the main governing equation being the energy conservation equation.

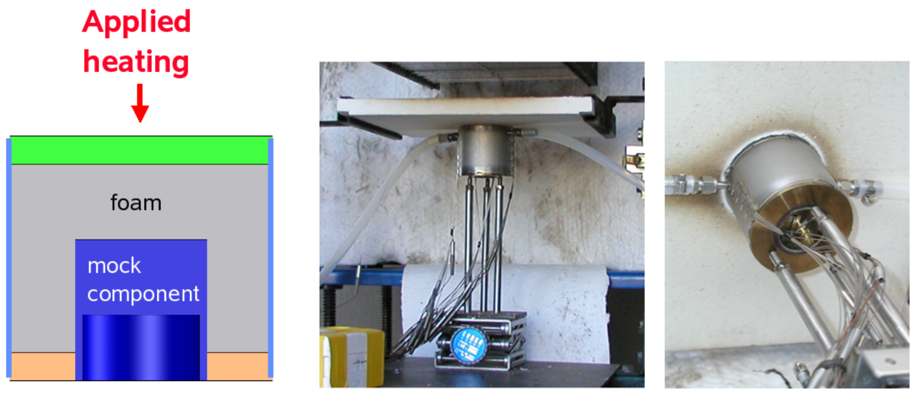


FIG. 3.1. *Illustrations of the "foam in a can" system*

The Calore thermal simulation has been configured to model the "foam in a can" experiment, but several of the simulation input parameters are still unknowns. In particular, we consider five calibration parameters: $q_2$, $q_3$, $q_4$, $q_5$, and $FPD$. The parameters $q_2$ through $q_5$ describe the applied heat flux boundary condition, which is not well-characterized in the experiments. The last calibration parameter, $FPD$, represents the foam final pore diameter, and is the parameter of most interest, because it will play a role in the ultimate modeling and prediction process. We want to consider the calibration of the Calore simulation for the temperature response up to 2200 seconds at nine different locations on the structure (six external and three internal).

**3.1. Preliminary analysis.** The first step is to collect a database of Calore simulation runs for different values of the calibration parameters. Ideally, we would like our design to provide good coverage for the posterior distribution of the calibration inputs. However, since we don't know the form of the posterior beforehand, we have to begin with an initial guess for the appropriate design.

Fortunately the method provides feedback, so if our original bounds are not adequate, they can be revised appropriately. This type of sequential approach has been discussed by [10] for Bayesian model calibration, and other examples are available in [2, 5, 1]. For our initial design, we used the DAKOTA software package to generate an LHS sample of size 50 using the variable bounds listed in Table 3.1.

Table 3.1
*Original design of computer experiments*

| Variable | Lower bound | Upper bound |
|---|---|---|
| $FPD$ | $2.0 \times 10^{-3}$ | $15.0 \times 10^{-3}$ |
| $q_2$ | 25,000 | 150,000 |
| $q_3$ | 100,000 | 220,000 |
| $q_4$ | 150,000 | 300,000 |
| $q_5$ | 50,000 | 220,000 |

The Bayesian calibration using these bounds illustrated that some adjustment to the bounds would be useful. Thus, we constructed a new LHS sample of size 50 using the revised design described in Table 3.2. The revised bounds are chosen so that they will cover the entire range of the posterior distribution for the calibration inputs.

Table 3.2
*Revised design of computer experiments*

| Variable | Lower bound | Upper bound |
|---|---|---|
| $FPD$ | $4.0 \times 10^{-3}$ | $6.0 \times 10^{-3}$ |
| $q_2$ | 25,000 | 150,000 |
| $q_3$ | 0 | 200,000 |
| $q_4$ | 100,000 | 400,000 |
| $q_5$ | 120,000 | 160,000 |

Using the results from the simulation runs, we can compare the ensemble of predicted time histories against the experimental time histories to see if the experimental data are "enveloped" by the simulation data. An example of this comparison is shown in Figures 3.2.

**3.2. Bayesian calibration analysis: nominal case.** Here we consider the Bayesian calibration using data from all nine "locations" of interest. Some of these "locations" (for example, location 1) are averages of multiple thermocouple readings, while others represent single thermocouple readings. For the analysis described here, the variance of $\varepsilon$ is not a function of time, and is the same at each location: the result is that all of the experimental data are weighed equally. However, the variance of $\varepsilon$ is still treated as an unknown, and is allowed to develop a posterior distribution along with the rest of the calibration parameters. For our prior distributions on the calibration parameters, we choose independent uniform distributions based on the bounds given in Table 3.1.

Each of the nine "locations" are modeled separately with two independent surrogates representing the response before and after 500 seconds, which results in a total of 18 surrogate models for the Calore output. We employ the use of multiple Gaussian process surrogate

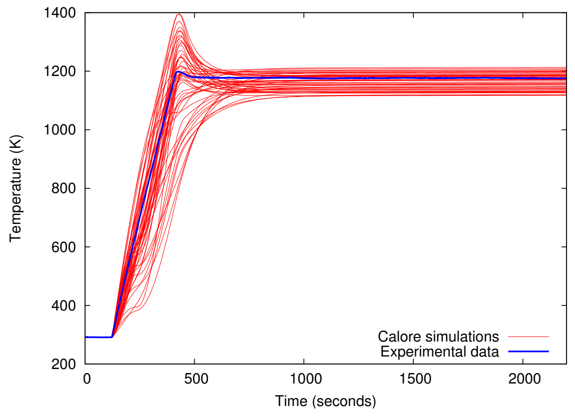


FIG. 3.2. *Location 1: Average lid temperature*

models because a single stationary Gaussian process representation of the response at all locations and time instances does not seem to be appropriate. Our choice of dividing the surrogates at 500 seconds is admittedly subjective (and a more comprehensive approach might choose different time divisions for different locations), but on average for the different locations, the process variance increases significantly around 500 seconds, and the correlation length with respect to time tends to increase as well for the latter part of the response.

For each surrogate, we employ an iterative algorithm to select an optimal subset of points with which to build the surrogate. At each location, the first surrogate is based on 75 points chosen optimally from the 1,950 available points (39 time instances $\times$ 50 LHS samples), while the second is based on 100 points chosen optimally from 8,550 points.

For the experimental data, we use 21 points evenly spaced at time intervals of 100 seconds for each of the 9 locations. The MCMC simulation is adjusted appropriately and run for 100,000 iterations. The resulting marginal posterior distributions for the parameter of interest, $FPD$, is shown in Figure 3.3.

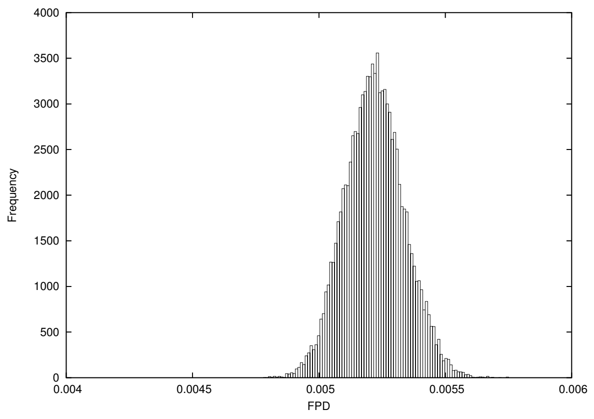


FIG. 3.3. *Histogram of posterior samples for $FPD$*

The statistics of the marginal posteriors are given in Table 3.3, and the pairwise correlation coefficients are given in Table 3.4. The correlation coefficients indicate a strong negative relationship between $q_2$ and $q_3$, as well as moderate negative relationships between $FPD$ and

$q_5$, and $q_3$ and $q_4$.

Table 3.3
*Posterior statistics based on the nominal calibration analysis*

| Variable | Mean | Std. Dev. |
|---|---|---|
| $FPD$ | $5.22 \times 10^{-3}$ | $1.17 \times 10^{-4}$ |
| $q_2$ | 88,546 | 16,977 |
| $q_3$ | 113,100 | 11,307 |
| $q_4$ | 246,270 | 11,652 |
| $q_5$ | 138,390 | 1,565 |

Table 3.4
*Pairwise correlation coefficients within the posterior distribution for the calibration of location 7*

| | $FPD$ | $q_2$ | $q_3$ | $q_4$ | $q_5$ |
|---|---|---|---|---|---|
| $FPD$ | 1.00 | 0.02 | 0.02 | -0.25 | -0.67 |
| $q_2$ | 0.02 | 1.00 | -0.80 | 0.18 | -0.02 |
| $q_3$ | 0.02 | -0.80 | 1.00 | -0.58 | -0.01 |
| $q_4$ | -0.25 | 0.18 | -0.58 | 1.00 | 0.00 |
| $q_5$ | -0.67 | -0.02 | -0.01 | 0.00 | 1.00 |

The total RMS agreement at all nine locations, based on the surrogate predictions at the posterior mean of the calibration inputs is 19.0. As a check on the surrogates, the total RMS based on the true Calore output, using the same inputs, is 18.9, which agrees well with that based on the surrogates. Finally, we note that agreement with the experimental data based on the posterior mean is *significantly* better than what was achieved using the global optimization algorithm DIRECT: after 65 runs of the Calore simulation, the best fit found by DIRECT had a total RMS of 32.3. Thus, the Bayesian approach used less total function evaluations than the least-squares point estimation approach (in which no surrogate was used for the optimization), found a better point estimate to the unknowns, and provides much more information about uncertainty present in the final solution.

**3.3. Adding prescribed input uncertainties.** In this section we extend the preceding analysis to include additional modeling uncertainties, as discussed in Section 2.2.1. While we have so far considered the calibration of five model inputs, there are actually many additional inputs to the Calore simulation which are subject to uncertainty or lack of knowledge. Here we study the effect on the calibration results when we treat thirteen additional model inputs as having prescribed uncertainties (in this case simply feasible bounds, represented by uniform probability density functions).

While it is also possible to treat these additional model inputs as calibration parameters, along with the original five, the primary reason for holding their uncertainties fixed is simply because there is an interest in knowing what effect this will have on the results. On the other hand, if they are treated as additional calibration parameters, their prior uncertainties may be reduced in light of the data $\boldsymbol{d}$, which would not give a picture of the effect of the prescribed uncertainties. Nevertheless, we conduct each of these analyses, as well as one "control" analysis, for comparison:

1. To make a fair comparison, we first conduct the analysis while holding the additional uncertain inputs fixed at their mean values. Although conceptually the same as the analysis discussed in Section 3.2, it is based on a different set of training data, and the

surrogates must now model the relationship between the additional thirteen inputs and the response, which we expect to result in additional overall uncertainty.

2. Using the method outlined in Section 2.2.1, we perform the analysis while allowing the additional inputs to vary according to their prescribed uncertainty distributions.
3. For comparison, we also perform the analysis in which the additional thirteen inputs are treated as calibration parameters, along with the original five.

The first step is to collect a new set Calore simulation data, which is necessary because the Gaussian process surrogates must now model the relationship between the temperature response and the thirteen new inputs, in addition to the five original calibration inputs. This results in a design of computer experiments over eighteen total variables, and surrogates that are based on nineteen inputs (since time is an input to the surrogates). We use a random LHS sample of size 50, with the bounds for the original parameters shown in Table 3.5 (for brevity, the information on the thirteen additional parameters is not shown). Generous bounds are used for the calibration parameters, since it is not known how much extra uncertainty will be introduced by the additional uncertain inputs.

TABLE 3.5
*Design of computer experiments for study with additional prescribed input uncertainties (specifications for additional thirteen inputs not listed)*

| Variable | Lower bound | Upper bound |
|---|---|---|
| $FPD$ | $2.0 \times 10^{-3}$ | $10.0 \times 10^{-3}$ |
| $q_2$ | 0 | 200,000 |
| $q_3$ | 0 | 200,000 |
| $q_4$ | 100,000 | 400,000 |
| $q_5$ | 50,000 | 200,000 |

With the new code runs, we use the same structure for our surrogates as before: two surrogates (for response before and after 500 seconds) are used at each of nine locations on the structure, for a total of eighteen surrogate models. We emphasize that the surrogates capture the temperature response as a function of time, the five original calibration inputs, and the thirteen additional uncertain inputs. We again employ the iterative point selection process, and this time between 40 and 128 points are used for each surrogate, depending on the complexity of the response.

Each of the three analyses described above are then conducted. For each case, we use 50,000 MCMC samples to construct the posterior. We note that these analyses are considerably more expensive than those described in Section 3.2. Of the three, the most expensive is the third case, in which the new inputs are treated as calibration inputs: the computational cost here is high because the MCMC sampler must evaluate the likelihood ratio (see Eq. (2.5)) once per iteration for each calibration input. Running on a Linux machine with a 64-bit, 2.4GHz processor, the third analysis took approximately 30 hours, while the first two took on the order of 10 hours each.

Since the calibration parameter $FPD$ is of most interest for the Calore simulation, we illustrate its posterior distribution in Figure 3.4, comparing each of the methods described above.

**4. Conclusions.** The important role that computational models play in prediction, design, and decision making necessitates appropriate methods for assessing the uncertainty in such predictions. This work has explored the use of Bayesian model calibration as a tool for calibrating a computational simulation with experimental observations, while at the same time keeping track of the uncertainty that is introduced in the process. We have also shown how

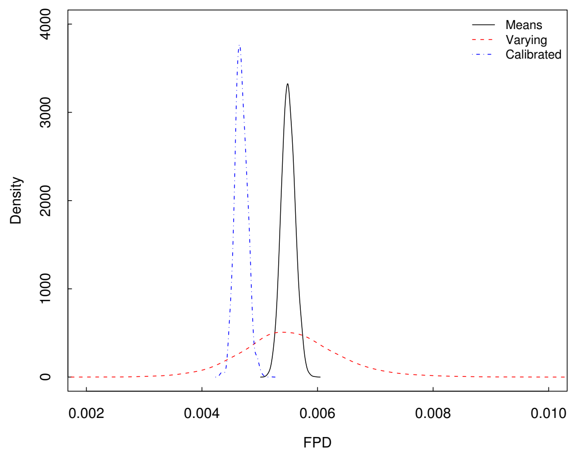


Fig. 3.4. *Comparison of posterior distribution of FPD for each of three approaches for treating the thirteen additional uncertain model inputs*

Gaussian process surrogate models can be used in place of an expensive simulation. Finally, we have developed a method which enables us to account for prescribed modeling uncertainties. We have applied this methodology to an expensive thermal simulation of "foam in a can" with a database of time-dependent experimental observations, and the results illustrate the promise of the methodology for uncertainty quantification and model calibration.

**Acknowledgment.** John McFarland would like to acknowledge helpful discussions with Youssef Marzouk regarding the theory underlying the Bayesian analysis of inverse problems (such as model calibration).

## REFERENCES

[1] R. Aslett, R. J. Buck, S. G. Duvall, J. Sacks, and W. J. Welch, *Circuit optimization via sequential computer experiments: design of an output buffer*, Applied Statistics, 47 (1998), pp. 31–48.
[2] M. C. Bernardo, R. J. Buck, L. Liu, W. A. Nazaret, J. Sacks, and W. J. Welch, *Integrated circuit design optimization using a sequential strategy*, IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems, 11 (1992), pp. 361–372.
[3] S. W. Bova, K. D. Copps, and C. K. Newman, *Calore: A computational heat transfer program, vol. 1: A theory manual and vol. 2, a user reference manual v. 4.3*, Sandia National Laboratories Technical Report, 2006-6083P (2006).
[4] S. Chib and E. Greenberg, *Understanding the Metropolis-Hastings algorithm*, American Statistician, 49 (1995), pp. 327–335.
[5] P. S. Craig, M. Goldstein, A. H. Seheult, and J. A. Smith, *Bayes linear strategies for matching hydrocarbon reservoir history*, in Bayesian Statistics 5, J. M. Bernardo, J. O. Berger, A. P. Dawid, and A. F. M. Smith, eds., Oxford University Press, Oxford, 1996, pp. 69–95.
[6] K. L. Erickson, S. M. Trujillo, J. B. Oelfke, C. R. Hanks, B. Belone, and D. M. Ramirez, *Component-scale removable epoxy foam thermal decomposition experiments part 1: Temperature data*, Sandia National Laboratories Technical Report, (2007, in revision).
[7] W. R. Gilks, S. Richardson, and D. J. Spiegelhalter, *Markov Chain Monte Carlo in Practice*, Chapman and Hall/CRC, Boca Raton, 1996.
[8] A. A. Giunta, J. M. McFarland, L. P. Swiler, and M. S. Eldred, *The promise and peril of uncertainty quantification using response surface approximations*, Structure and Infrastructure Engineering, 2 (2006), pp. 175–189.

[9] W. K. Hastings, *Monte Carlo sampling methods using Markov chains and their applications*, Biometrika, 57 (1970), pp. 97–109.

[10] M. C. Kennedy and A. O'Hagan, *Bayesian calibration of computer models*, Journal of the Royal Statistical Society B, 63 (2001).

[11] N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, and E. Teller, *Equations of state calculations by fast computing machines*, Journal of Chemical Physics, 21 (1953), pp. 1087–1092.

[12] C. Rasmussen, *Evaluation of Gaussian processes and other methods for non-linear regression*, PhD thesis, University of Toronto, 1996.

[13] V. J. Romero, J. W. Shelton, and M. P. Sherman, *Modeling boundary conditions and thermocouple response in a thermal experiment*, ASME paper IMECE2006-15046, 2006 International Mechanical Engineering Congress and Exposition, Nov. 5-10, Chicago, IL (2006).

[14] A. Vecchia and R. Cooley, *Simultaneous confidence and prediction intervals for nonlinear regression models, with application to a groundwater flow model*, Water Resources Research, 23 (1987), pp. 1237–1250.

# MESH OPTIMIZATION FOR CURVED DOMAINS WITH TARGET-MATRIX PARADIGM

HALE ERTEN[†] AND PATRICK M. KNUPP[‡]

**Abstract.** In order to have more accurate discretization models for physical domains with curved boundaries, we address an important problem involving meshes on curved boundaries. Our method uses advantages of recently introduced “target-matrix” paradigm and mesh optimization techniques to provide quality meshes which are depending on relative size and ideal shape constraints. The proposed approach is applied to linear triangular elements. Possible extensions to the quadratic-element case for two-dimensional domains are discussed. Examples illustrate the behavior of the method whose goal is to untangle initial meshes near the boundary, as well as to produce good quality elements. The improvement approach involves node-movement by optimizing the objective function based on a non-barrier shape and size metric. We offer thoughts on using this procedure for real life applications in three-dimensions.

## 1. Introduction.

Finite element methods have been widely used throughout the years in engineering simulations. A major requirement is that the discretization should represent the physical domain accurately. Higher-order elements can be used, such as those discussed in [10], to overcome this problem which can be critical for many applications. This approach requires appropriate mesh quality on the boundary, where the high-order elements are often located. In the next section we formulate the basic problem. This is followed by a brief literature survey, and a description of our proposed approach for the linear element case. Results for linear three-point triangular element meshes will be given, and possible extensions to the quadratic case will be discussed.

### 1.1. Problem Definition.

We describe the basic problem in Figure 1.1. As can be seen from the figure, an element edge outside physical boundaries is likely to happen in both two and three-dimensional meshes which can lead to inaccurate results or problems during finite element calculations.

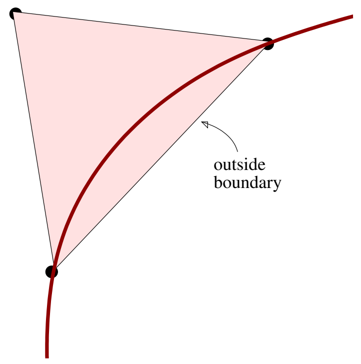


FIG. 1.1. *Around boundary region of a computed triangular mesh, an element whose edge is outside the physical boundary.*

Having an element outside the geometric boundary, is more likely when we have curved domains. In order to solve this problem, a detection method is needed for finding poor quality elements. Once these are found, there should be methods to fix these elements. For the detection part, the first thing that occurs to us is to use the determinant of the Jacobian of

[†]University of Florida Computer Info. and Sciences & Engineering, herten@cise.ufl.edu
[‡]Sandia National Laboratories, pknupp@sandia.gov

the map on each element, satisfying positivity at Gauss points of the element. If we have an invalid element based on this detection criteria, the following operations could be considered to fix them: *(a)* swap(edge/element) *(b)* h-refinement *(c)* curving *(d)* collapsing *(e)* moving nodes. These operations were considered in solving a similar problem in [7, 3, 8]. In this paper we focus exclusively on the latter technique because it has not been investigated as carefully as the other operations. Untangling higher-order elements are a critical part of the problem and will be considered first.

**1.2. Previous Work.** For curved boundary domains, generating meshes having quality constraints requires additional effort to have better accuracy for finite element analysis applications. Boundary-intersecting elements can lead to inaccurate results in finite element method. For curved domains, different kinds of approaches have recently emerged. Shephard et al. investigate this issue in several papers [7, 3, 8], focusing on p-version mesh generation for curved domains mostly in three-dimensional meshes. In [3], not only detecting element interferences due to curved features of the elements are discussed, but also element-modifying methods for obtaining valid curvilinear meshes for applications are introduced.

Apart from using the Jacobian matrix properties, [7, 3, 8] considers the benefits of using Bezier curves for element edges instead of normal quadratic interpolations. Problematic cases are described and for each of them, a suitable solution is proposed, like edge swapping, face swapping, edge deletion, etc. Vertex relocation is not considered in depth in these references.

Branets and Carey studied a related problem from a metric point of view. They recently introduced a local quality metric [1], along with a smoothing algorithm. It has been modified for elements having curved boundary edge or surface [2]. Again, the Jacobian matrix has been used as a part of the quality metric representing a linear combination of shape distortion and dilation metric. For invalid elements, the metric has been extended by setting its value to infinity for those elements. Most importantly, [2] gives necessary conditions to have non-degenerate mapping for two and three-dimensional triangular and tetrahedral quadratic elements respectively as well as similar discussions about quadrilateral meshes. The new can help to find degeneracies and element distortion. The study does not consider the case of initially tangled meshes.

Aside from the above-mentioned studies, quadratic curved meshes for curved boundary domains are little considered in the literature (but see [10, 9]).

There are several quality metrics which can be adapted to this problem to eliminate invalid elements by local or global optimization techniques. Knupp showed that quality metrics can be represented algebraically depending on the Jacobian matrix of the elements [5]. Continuing this algebraic approach, same author introduced the "target-matrix" paradigm for mesh optimization [4]. By using target, active, and weighted-active matrices, mesh optimization under an objective function can be very effective. Important concepts, such as being a barrier or a non-barrier metric, are defined and their essential behaviors were discussed in [6] while introducing local two-dimensional metrics to be used in objective functions.

**2. Proposed Solution.** Our method is based on the idea of relocating $free$ element vertices depending on an objective function. The first thing we need to consider is choosing the appropriate metric on which to base our objective function. Apart from the objective function, our method of detecting invalid elements and deciding which vertices are to be $free$ is critical to the success of the method. As mentioned in the previous section, the determinant of the Jacobian Matrix should be positive ($\det J > 0$) in order to have *valid* or untangled elements. For linear finite element case, the determinant will be constant on simplicial elements, but for the quadratic element case, $\det J$ varies, even for simplicial elements. Thus, it is more difficult to decide whether or not a quadratic element is valid. One way to do this is to require that $\det J$ be positive at all the Gauss points of the element or, more generally, at selected *sample* *points* within the element. Another important consideration is the question of which nodes and vertices within an element ones should be considered $free$ nodes and which should be fixed. Since we focus on the boundary, moving mid-side nodes and corner vertices not owned by the boundary can be a suitable solution. There are many combinations to consider, but since this work is intended as only an initial exploration, this question will be studied later. For now, we will mainly focus on trying to explore the behavior of the method by applying it on linear elements.

**2.1. Metric.** We have noted that, second author described the use of "target-matrix" paradigm in [4] while mentioning the advantages of using algebraic properties of mesh elements. In the continuation of this work [6], two-dimensional metrics have been used to construct objective functions for mesh optimization. Their characteristic properties are pointed out and the difference between barrier and non-barrier metrics was emphasized.

For our problem, we can first consider untanglers, like (2.1) or (2.2). But, it is known that these do not always work, although they do relatively well on local patches, which could be the case if one is interested in improving a local mesh near the boundary. The objective functions are

$$\min\ F = \sum_k (|\alpha_k| - \alpha_k) \geq 0, \tag{2.1}$$

$$\max\{\min\ \alpha_k\}, \tag{2.2}$$

where $A$ : Jacobian Matrix, $\alpha = \det A$, and $k$ is an element index which runs over a local patch of elements. However, these metrics do not improve shape-quality.

We can also consider a metric which improves shape, like mean ratio (2.3), but this metric requires the initial mesh to be untangled.

$$\min\ F = \sum_k \frac{||A_k||_F^2}{2 * \alpha_k}\ if\ \alpha_k > 0, \tag{2.3}$$

Because the boundary mesh is likely to be tangled, we will instead use a non-barrier form of the shape and size metric as follows:

$$\mu = ||T||_F^2 - \sqrt{||T||_F^2 + 2 * \tau} + 2, \tag{2.4}$$

where $T = A * W^{-1}$, $\tau = \det T$, and $||\cdot||_F$ is the Euclidean (Frobenius) matrix norm [6]. With an appropriately constructed target matrix, this metric should make it possible to both untangle and improve the shape quality of the boundary elements.

In this metric, note that $W$ can vary from one element to another, according to the size of the element in the initial mesh. It has the interval from 0, for ideal element, to $\infty$, for highly invalid element; so that optimization will try to reach minimum value which is 0. If the minimum is attained, $A = RW$, i.e., the Jacobian matrix within the physical element will equal the target matrix, up to an arbitrary rotation.

As a review, Figure 2.1 shows the target, active and weight matrices in the Target Matrix paradigm.

**2.2. Method.** We begin by constructing an appropriate target matrix for each element. Basically, we will start with a $W$ based on an ideal element with unit length, that we will scale according to size of the physical element. Since we are scaling the target matrix based on area, this will produce elements sized relative to their initial sizes in optimized mesh. If

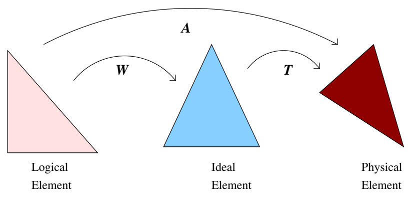


FIG. 2.1. *The relationship between target, active and weight matrices in terms of mapping between elements.*

an application needs differently sized elements in different regions of the domain, the scaled target will preserve this initial mesh heterogeneity even after the optimization process.

After calculating suitable target matrices, metric value for each element should be computed to produce objective value to be minimized. That is, optimization algorithms can be used to optimize the locations of free points to have better shaped and sized elements.

Notice that, the scaling factor for target matrix can be selected by different schemes. Following section will illustrate the differences of proposed three different schemes as well as show some examples of above mentioned optimization algorithm.

**2.3. Schemes and Results.** One of the most important parts of our method is having an appropriate target matrix, which can be done using many schemes. We describe three reasonable construction schemes in detail.

**2.3.1. Target Construction Algorithms.** We begin with a general $W$ matrix which represents the mapping between right triangle to a unit equilateral triangle. This can be considered as the mapping from the logical to the ideal element.

The steps for scaling $W$ can be given as follows:

- Set

$$W = \begin{pmatrix} 1 & 0.5 \\ 0 & \sqrt{3}/2 \end{pmatrix}$$

- $W' = c * W$, where $c$ : *constant*

$$c = \frac{2 * \sqrt{Area_i}}{3^{1/4}} \tag{2.5}$$

$$c = \frac{2 * \sqrt{\det A_i/2}}{3^{1/4}} \tag{2.6}$$

We can rewrite above relation in terms of determinants:

$$\det W' = c^2 * \det W = \frac{4 * Area_i}{3^{1/2}} \tag{2.7}$$

- $Area_i$ : Area of the element $i$, which can be calculated using the three vertices of the triangle as follows:

$$Area(\triangle) = \sqrt{s(s-a)(s-b)(s-c)}, \tag{2.8}$$

where $s = \frac{1}{2}(a+b+c)$ and $a, b, c$ : found by Euclidean distance formula from the vertices of the edges

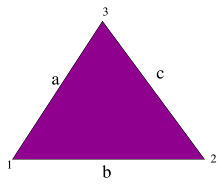


FIG. 2.2. *Simple triangle.*

This area calculation always gives positive values, but when our initial mesh has inverted elements, we need to be aware of that situation. Therefore, signed area calculation must be made which is exactly equivalent to above area calculation except signed information. In other words,

$$Area(\triangle) = |\det(\triangle)|/2 \tag{2.9}$$

As we have mentioned earlier, initially tangled meshes must be detected and our approach must untangle the elements. Several different schemes for this goal can be introduced, three logical and efficient schemes are given. Remember that after finding the $Area_i$ for each element, we will be finding the corresponding update constant $c$ to modify the unscaled $W$ appropriately.

- Possible schemes for $c$:

(I) **Absolute Value Area**

$Area_i$ : calculate from 3-side lengths $\geq 0$

$$Area_i = |\det A_i|/2 \tag{2.10}$$

Overall, it will be

$$Area_i = \begin{cases} (-1) * \det A_i/2, & \text{if } \det A_i < 0 \\ \det A_i/2, & \text{otherwise} \end{cases} \tag{2.11}$$

This approach has the advantage of being a quick fix to the problem and gives relative size of the untangled version, since at least it has some information from the topology. However, it is not exactly correct, because some areas are counted twice or more by this way. This scheme has another drawback which is depicted as Example #4 in the next section, such that positive area can lead to larger elements than mesh boundaries causing tangled elements.

(II) **Global Average Area**

$Area_i$ : average element area

$$Area_i = \frac{A_{total}}{N}, \text{ if } \det A_i < 0 \tag{2.12}$$

Overall, it will be

$$Area_i = \begin{cases} \frac{A_{total}}{N}, & \text{if } \det A_i < 0 \\ \det A_i/2, & \text{otherwise} \end{cases} \tag{2.13}$$

Note that, while calculating $A_{total}$, we will be using signed area ($\det A_i/2$). If the mesh is nearly structured, this method can be a good idea, since it will give some information about overall topology. On the other hand, the mesh can have a tangled area having large triangles, whereas the average area is too small. Although it is not usually possible, this can yield skinny elements at the end when we untangled the mesh, but the above-mentioned problem about Example #4, will not appear.

(III) **Patch Average Area**

$Area_i$ : average element area based on patch of the inverted element

$$Area_i = \frac{A_{patchTotal}}{N_{patch}}, \text{ if } \det A_i < 0 \tag{2.14}$$

Overall, it will be

$$Area_i = \begin{cases} \frac{A_{patchTotal}}{N_{patch}}, & \text{if } \det A_i < 0 \\ \det A_i/2, & \text{otherwise} \end{cases} \tag{2.15}$$

Note that, while calculating $A_{patchTotal}$, we will be using signed area ($\det A_i/2$).

For an inverted element, find the connected patch of each corner points and after calculating their signed area, compute the average area value. This will be used for determining target matrix $W$. This is more complicated, however it can give better results in comparison to first approach.

- Pseudo-code for general algorithm:
  *Input:* Initial Mesh Elements
  *Output:* $W$ matrix for each element

```
Begin
  Set
  $W = \begin{pmatrix} 1 & 0.5 \\ 0 & \sqrt{3}/2 \end{pmatrix}$
  totalArea = 0
  Foreach Triangular Element $i$
    Find $\det A_i$, record in an array
    $totalArea+ = \det A_i/2$
  End Foreach
  Foreach Triangular Element $i$
    if $\det A_i < 0$
    then $c = \frac{2*\sqrt{(-1)*\det A_i/2}}{3^{1/4}}$ // for absolute value scheme
        or $c = \frac{2*\sqrt{totalArea/numberOfElements}}{3^{1/4}}$ // for global average scheme
        or find patches of the corners
        calculate average element area in these patches
        $c = \frac{2*\sqrt{averagePatchArea_i}}{3^{1/4}}$ // for average patch area scheme
    else
        $c = \frac{2*\sqrt{\det A_i/2}}{3^{1/4}}$
  End Foreach
$W_i = c * W$
End
```

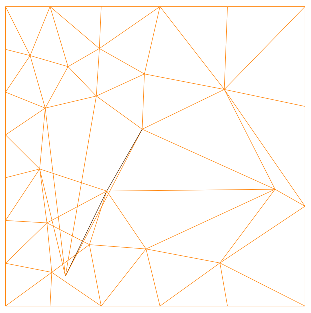


FIG. 2.3. *Input data set (Example #1)*

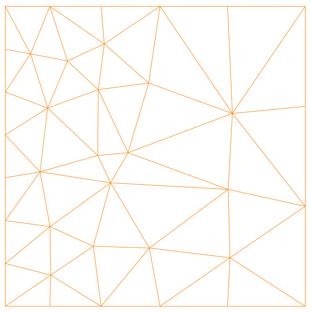


FIG. 2.4. *Absolute Value (Example #1)*

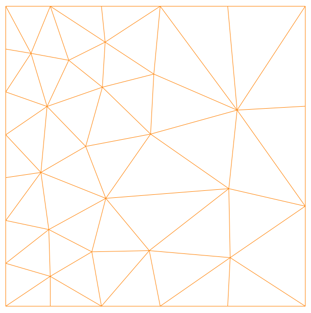


FIG. 2.5. *Global Average (Example #1)*

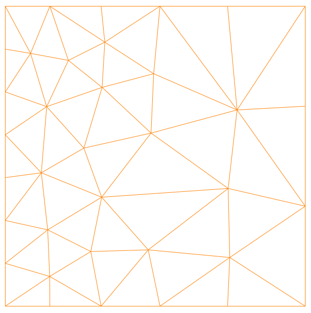


FIG. 2.6. *Patch Average (Example #1)*

We should point out that this algorithm works for both tangled and untangled initial meshes. The former case is just a special case of the latter.

**2.3.2. Examples.** In this section, examples depict the differences between the three above mentioned target metric construction methods. Tables showing various quality metric value allow comparisons between the initial mesh and resulting meshes after optimization using corresponding target matrix construction schemes. The quality metrics in the Table are: the average, maximum and minimum element area values and minimum element angle degrees as well as initial and final objective function values after optimization process.

**Example #1:** *Unstructured mesh in a square domain (Containing tangled elements)* Our metric successfully untangles the mesh and optimizes according to given schemes as in figures. Statistics related to initial and final meshes can be seen from the table. (Figure 2.3, Figure 2.4, Figure 2.5, Figure 2.6; Table 4.1)

**Example #2:** *Unstructured mesh with curved boundary domain (Containing NO tangled elements)* Our metric successfully optimizes the mesh according to given absolute value area scheme. Since there is no inverted element whose det $J < 0$, therefore we do not need to run the other schemes which will give the same resulting mesh. (Figure 2.7, Figure 2.8; Table 4.2)

**Example #3:** *Unstructured mesh with curved boundary domain (Containing tangled elements)* Our metric successfully untangles the mesh and optimizes the mesh according to given schemes. (Figure 2.9, Figure 2.10, Figure 2.11, Figure 2.12; Table 4.3)

**Example #4:** *Unstructured mesh with curved boundary domain (Extremely inverted elements)* When we have an extremely bad inverted element, our metric does not succeed in untangling the initial mesh under first and third schemes. The reason may possibly be because the inverted element areas are very large, resulting in large values in the target matrix $W$; then

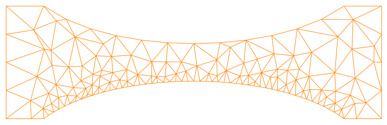


FIG. 2.7. *Input data set (Example #2)*

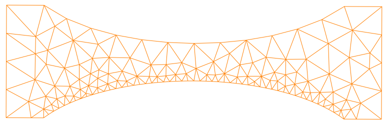


FIG. 2.8. *Absolute Value (Example #2)*

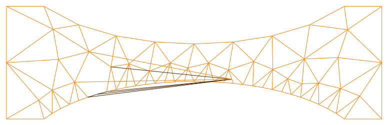


FIG. 2.9. *Input data set (Example #3)*

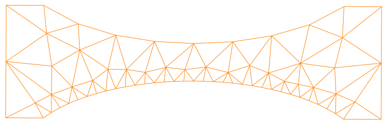


FIG. 2.10. *Absolute Value (Example #3)*

optimization based on the target (which is poorly chosen) will again make the elements tangled. The relative ratio between the untangled element areas and the tangled element areas is significant; that will result in failure of the untangling step. The second scheme succeeds because it considers the total polygonal area when we have a valid initial mesh. (Figure 2.13, [Close-ups] Figure 2.14, Figure 2.15, Figure 2.16; Table 4.4)

**3. Quadratic Elements.** We have seen from the examples above, our metric works more or less as expected for the linear element case. Target construction is critical to success. To extend this approach to the case of quadratic elements, our method will include *sample points*, defined within the master element, whose purpose is to sample the local quality at various points within the physical element. It is expected that, by attempting to improve quality at all the sample points, problematic quadratic elements near the boundary will be eliminated. Another important ingredient in this approach will be to determine which nodes and vertices within an element should be permitted to be $free$. There are many potential combinations to consider before proposing a final algorithm.

Although, this work does not contain results about the quadratic case, we expect similar behavior as with the linear case. In order to have a more efficient algorithm, we can use linear elements inside regions. In other words, just the boundary-related elements can be defined with six nodes, whereas the others can just be ordinary three-node triangles. This will naturally require different cases to be handled in algorithm, since hybrid mesh elements can affect each other in several ways.

**4. Future Work.** Most of the geometric models contain curved boundaries that need to be handled in a more careful way in terms of the mesh. We have introduced preliminary results towards having quality meshes in such domains by using a “target-matrix” paradigm-inspired objective function. In the future, we want to explore our method in quadratic triangular meshes in two dimensions and quadratic tetrahedra in three dimensions. Additionally, a similar approach could perhaps be used to optimize quadratic quadrilateral mesh elements as well as their three-dimensional hexahedral counterparts. As the next step our method's behavior on quadratic two-dimensional triangle elements should be understood clearly to be able to continue to three-dimensional mesh elements. Note that, the later case will have more degrees of freedom as well as additional cases, such as surface interferences in addition to edge boundary conflicts in two-dimensional case.

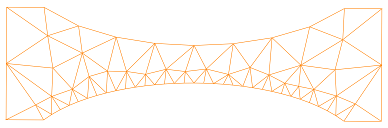


FIG. 2.11. *Global Average (Example #3)*

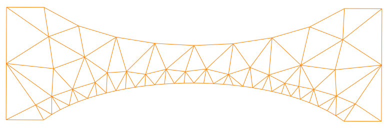


FIG. 2.12. *Patch Average (Example #3)*

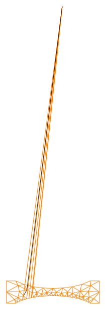


FIG. 2.13. *Input data set (Example #4)*

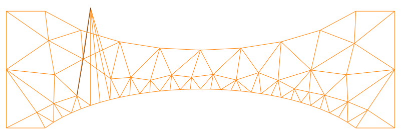


FIG. 2.14. *Absolute Value (Example #4)*

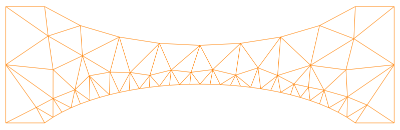


FIG. 2.15. *Global Average (Example #4)*

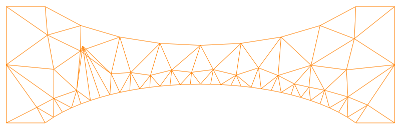


FIG. 2.16. *Patch Average (Example #4)*

TABLE 4.1
*Comparison of different schemes for Example #1*

| **Area** | Initial | Absolute | Global Average | Patch Average |
|---|---|---|---|---|
| Average | 0.0192308 | 0.0192308 | 0.0192308 | 0.0192308 |
| Max | 0.057672 | 0.046239 | 0.0448294 | 0.0447587 |
| Min | -0.0368821 | 0.00480185 | 0.0060206 | 0.00602475 |
| **Min Angle** | Initial | Absolute | Global Average | Patch Average |
| Average | 38.7463 | 41.9124 | 43.6603 | 43.6659 |
| Max | 57.2982 | 56.7738 | 57.8124 | 57.8742 |
| Min | 1.67618 | 16.9158 | 28.2249 | 28.219 |
| **Obj.Func.Val.** | 64.884 | 11.4389 | 9.67038 | 9.69442 |

TABLE 4.2
*Comparison of different schemes for Example #2*

| **Area** | Initial | Absolute | Global Average | Patch Average |
|---|---|---|---|---|
| Average | 0.0007667 | 0.0007667 | 0.0007667 | 0.0007667 |
| Max | 0.00381743 | 0.00417208 | 0.00417208 | 0.00417208 |
| Min | 5.09701e-05 | 7.02992e-05 | 7.02992e-05 | 7.02992e-05 |
| **Min Angle** | Initial | Absolute | Global Average | Patch Average |
| Average | 39.9638 | 44.4628 | 44.4628 | 44.4628 |
| Max | 58.6392 | 57.7607 | 57.7607 | 57.7607 |
| Min | 11.0711 | 26.9858 | 26.9858 | 26.9858 |
| **Obj.Func.Val.** | 56.3769 | 33.1737 | 33.1737 | 33.1737 |

TABLE 4.3
*Comparison of different schemes for Example #3*

| **Area** | Initial | Absolute | Global Average | Patch Average |
|---|---|---|---|---|
| Average | 0.00199746 | 0.00199746 | 0.00199746 | 0.00199746 |
| Max | 0.0168338 | 0.00810362 | 0.00824869 | 0.00844283 |
| Min | -0.0107991 | 0.000352017 | 0.000352017 | 0.000352017 |
| **Min Angle** | Initial | Absolute | Global Average | Patch Average |
| Average | 35.2685 | 39.8439 | 39.6595 | 40.045 |
| Max | 57.6292 | 59.4528 | 59.4528 | 59.4528 |
| Min | 0.689214 | 18.8607 | 18.86 | 18.8555 |
| **Obj.Func.Val.** | 197.369 | 20.8239 | 20.2379 | 20.0906 |

TABLE 4.4
*Comparison of different schemes for Example #4*

| **Area** | Initial | Absolute | Global Average | Patch Average |
|---|---|---|---|---|
| Average | 0.00199746 | 0.00199746 | 0.00199746 | 0.00199746 |
| Max | 0.0768827 | 0.00817313 | 0.0081784 | 0.00796626 |
| Min | -0.12003 | -0.0051679 | 0.000352017 | -0.000668633 |
| **Min Angle** | Initial | Absolute | Global Average | Patch Average |
| Average | 34.9136 | 38.0154 | 39.6869 | 38.0637 |
| Max | 57.6292 | 59.4528 | 59.4528 | 59.4528 |
| Min | 0.0882529 | 0.705251 | 18.9284 | 2.3893 |
| **Obj.Func.Val.** | 1329.18 | 24.0698 | 23.3629 | 23.90.12 |

## REFERENCES

[1] L. BRANETS AND G. F. CAREY, *A local cell quality metric and variational grid smoothing algorithm*, Procedings of the 12th International Meshing Roundtable, (2003), pp. 371–378.

[2] ———, *Extension of a mesh quality metric for elements with a curved boundary edge or surface*, Journal of Computing and Information Science in Engineering, 5 (2005), pp. 302–308.

[3] S. DEY, R. M. O'BARA, AND M. S. SHEPHARD, *Towards curvilinear meashing in 3d: The case of quadratic simplices*, Computer Aided Design, 33 (2001), pp. 199–209.

[4] P. M. KNUPP, *Formulation of a target-matrix paradigm for mesh optimization*, Submitted.

[5] ———, *Algebraic mesh quality metrics*, SIAM Journal on Scientific Computing, 23 (2001), pp. 193–218.

[6] P. M. KNUPP AND U. HETMANIUK, *Local 2d metrics for mesh optimization in the target-matrix paradigm*, Submitted.

[7] X.-J. LUO, M. S. SHEPHARD, R. M. O'BARA, R. NASTASIA, AND M. W. BEALL, *Automatic p-version mesh generation for curved domains*, Engineering with Computers, 20 (2004), pp. 273–285.

[8] X.-J. LUO, M. S. SHEPHARD, AND J.-F. REMACLE, *p-version mesh generation issues*, Proceeding of the 11th International Meshing Roundtable, (2002), pp. 343–354.

[9] M. S. S. SAIKAT DEY AND J. E. FLAHERTY, *Geometry representation issues associated with p-version finite element computations*, Computer Methods in Applied Mechanics and Engineering, 150 (1997), pp. 39–55.

[10] S. SHERWIN AND J. PEIRO, *Mesh generation in curvilinear domains using high-order elements*, Intenational Journal for Numerical Methods in Engineering, 53 (2002), pp. 207–223.

# A METHOD OF MANUFACTURED SOLUTIONS FOR PDES WITH STOCHASTIC INPUTS

PAUL CONSTANTINE[*] AND PATRICK M. KNUPP [†]

**Abstract.** Partial differential equations with stochastic inputs have become popular for modeling engineering systems with uncertain input parameters. As a result of this popularity, the uncertainty quantification community has developed powerful numerical techniques to approximate statistics of the solutions to these systems. However, very little work has gone into verifying the computation of these statistics. In this paper, we apply the method of manufactured solutions – a technique for verifying the convergence rate of the numerical solution of a deterministic system – to the stochastic case in order to verify the convergence of the computed statistics.

**1. Introduction.** In many – if not most – modern engineering applications, the quantities determining the behavior of the engineering system are known with only a relative degree of certainty. Instead of trying to account for these uncertainties in the models, the typical practitioner will simply choose to use mean quantities or impose safety factors for critical parameters. Recently, in the search for more realistic and accurate models, many researchers have used a probabilistic framework to incorporate the uncertainties. One common approach is to represent the uncertain inputs as random processes and replace the deterministic parameters in the existing modeling equations. These uncertain inputs may include boundary conditions, domain specifications, forcing terms, or equation parameters describing material properties. By introducing randomness into the models, the solution to the governing equations becomes random itself. Then the objective is to quantify the uncertainty in the random solution. This typically involves computing approximate moments of the solution, though in some cases one may want an approximate probability density function and information derived from that, e.g. the probability that a quantity of interest exceeds some critical value. In this paper we focus on computation of the moments.

There are a number of existing techniques for approximating the moments. The simplest is a Monte Carlo approach, where the random quantities are sampled according to their distribution. For each realization of the random quantity, a deterministic system is solved. The results from the uncoupled deterministic solutions are then post-processed to compute the approximate moments. One technique that has risen to prominence in the uncertainty quantification community is the stochastic Galerkin technique [2]. This technique employs a spectral representation of the random quantities and projects the governing equations onto a finite set of basis functions, much like the deterministic Galerkin technique. The projection creates a set of coupled deterministic equations for the coefficients of the spectral expansion of the solution. The moments are then explicit expressions of the computed coefficients. Another technique for approximating the moments, known as stochastic collocation [5], uses quadrature rules to approximate the multidimensional integrals in probability space that define the moments. We review these techniques in further detail in section 2.

Each of these techniques has a solid theoretical foundation with analytically determined convergence rates: the convergence rate of Monte Carlo is proportional to the square root of the number of samples, the spectral expansion of the random process in stochastic Galerkin converges to the solution exponentially as the number of terms in the series increases, and the stochastic collocation approximations converge to the exact integrals exponentially as the number of points in the quadrature rule increases. The literature describing these techniques typically demonstrates their convergence rates by applying the technique to a simple problem with a known solution and producing convergence plots. Most real problems, however,

[*]Stanford University, paul.constantine@stanford.edu
[†]Sandia National Laboratories, pknupp@sandia.gov

do not have a known solution. Given results from a code implementing one of these techniques, how can one be sure that the results are correct, i.e. that there are no mistakes in the implementation? For complex problems, it is common practice to compare the results from different techniques to one another; if the results converge to the same answer, then the codes are verified. We propose a better way.

The method of manufactured solutions [8] is a technique for verifying the convergence of a given implementation. It has been applied to deterministic problems where convergence is defined as the approach of the numerical solution to the exact solution as the spatial and temporal step sizes go to zero. The essence of the technique is to apply the operator of a given PDE to an arbitrary – or "manufactured" – function. This produces a right hand side for the PDE which is plugged in to the solver. With an analytically manufactured solution available, one can perform a grid convergence study with the given code to verify the rate of convergence. We describe this method in more detail in section 3.

The main contribution of this paper is the application of the method of manufactured solutions to PDEs with stochastic inputs. We devise verification methods for implementations of the techniques mentioned above that compute moments of the stochastic solutions. The idea is the same as the deterministic case: apply the stochastic differential operator to an arbitrary function producing a stochastic right hand side and perform a convergence study on the given implementation. We describe the methods in greater detail and give three demonstrative examples in section 4.

## 2. Uncertainty Quantification Techniques.

In this section we describe in further detail the three techniques mentioned above for computing the moments of the random solution of the PDE with random inputs. Each of these techniques has its advantages and disadvantages. For the sake of comparison and clarity, we present each technique in the context of a model problem.

Let $\mathcal{D} \subset \mathbb{R}^n$ be a spatial domain, and define a complete probability space $(\Omega, \mathcal{F}, \mathbf{P})$. Let $\boldsymbol{\xi}$ be a vector of i.i.d. random variables with joint probability density function $W_{\boldsymbol{\xi}}$, and assume that $\boldsymbol{\xi}$ is measurable with respect to $(\Omega, \mathcal{F}, \mathbf{P})$. Let $\alpha(\mathbf{x}, \boldsymbol{\xi})$ be an $L_2$ spatially varying random process defined on $\mathcal{D} \times \Omega$ such that $\alpha(\mathbf{x}, \boldsymbol{\xi}) > 0$ for all $\mathbf{x} \in \mathcal{D}$ and all possible values of $\boldsymbol{\xi}$. The model problem is given by the elliptic equation

$$\nabla \cdot (\alpha(\mathbf{x}, \boldsymbol{\xi}) \nabla u(\mathbf{x}, \boldsymbol{\xi})) = f \qquad \mathbf{x} \in \mathcal{D}$$

with boundary conditions $u = 0 \in \partial\mathcal{D}$. Note that the solution $u(\mathbf{x}, \boldsymbol{\xi})$ is a random process. The objective is then to compute $\mathrm{E}[u(\mathbf{x}, \boldsymbol{\xi})]$ and $\mathrm{Var}[u(\mathbf{x}, \boldsymbol{\xi})]$.

### 2.1. Monte Carlo.

The Monte Carlo approach is very appealing for its simplicity and ease of implementation. First choose the number of samples $M$. Then for $j = 1 \dots M$, choose a sample $\boldsymbol{\xi}_j$ from the distribution of $\boldsymbol{\xi}$. With this sample, compute $\alpha_j = \alpha(\mathbf{x}, \boldsymbol{\xi}_j)$. Next solve the deterministic problem

$$\nabla \cdot (\alpha_j(\mathbf{x}) \nabla u_j(\mathbf{x})) = f \qquad \mathbf{x} \in \mathcal{D}.$$

Now with the $M$ solutions $u_j$, one can compute unbiased approximations to the moments [7] as

$$\mathrm{E}[u](x) \approx \frac{1}{M} \sum_{j=1}^{M} u_j(x) \equiv \mu(x) \qquad \mathrm{Var}[u](x) \approx \frac{1}{M-1} \sum_{j=1}^{M} (u_j(x) - \mu(x))^2$$

One of the primary advantages of this approach is that it is *non-intrusive*. In other words, this approach will work with a given *deterministic* solver without altering the code. It simply uses multiple runs of the existing deterministic solver with varying values of the input parameters. Thus, if the deterministic solver is trusted, then debugging the Monte Carlo approach is relatively straightforward. The primary drawback to this method is the slow convergence rate proportional to $\sqrt{M}$. If the deterministic solver takes weeks to compute a single solution, then using $M = 10,000$ samples for two significant digits of accuracy in the approximate expectation is quite unrealistic. There have been attempts to accelerate convergence by "intelligently" sampling the distributions of the random inputs – such as Latin Hypercube and Hammersley sampling techniques – but fundamentally these still suffer from the same drawback.

**2.2. Stochastic Galerkin.** The stochastic Galerkin technique has its roots in the work of Wiener [10] who expressed an arbitrary Gaussian process as an infinite series of orthogonal Hermite polynomials. The polynomials in this series take a vector of Gaussian random variables as arguments. Ghanem and Spanos [2] utilized a truncated version of this representation in their stochastic finite element method, which they applied to structural mechanics problems with Gaussian inputs. In 2002, Xiu and Karniadakis [11] extended this method to non-Gaussian processes by incorporating the Wiener-Askey family of orthogonal polynomials.

The first step of this method is to represent the random input parameters using a vector of random variables. If the input is a spatially varying random field, this can involve another orthogonal expansion - the Karhunen-Loeve expansion [4] - which decomposes the process into a series based on the eigenpairs of the covariance kernel and a countable set of random variables. For our model problem, we assume that $\alpha$ is an explicit function of a vector of random variables $\boldsymbol{\xi}$, thus avoiding the need for such an expansion.

The next step is to express the solution $u$ as series expansion with a chosen set of orthogonal basis functions $\{\Psi(\boldsymbol{\xi})\}$ and truncate this series at some predetermined number of terms $N$:

$$u(\mathbf{x}, \boldsymbol{\xi}) = \sum_{i=0}^{N} u_i(\mathbf{x}) \Psi_i(\boldsymbol{\xi})$$

The number $N$ is a function of the number $n$ of components of $\boldsymbol{\xi}$ and the highest degree of basis polynomial $m$:

$$N + 1 = \frac{(n+m)!}{n!m!}.$$

The polynomials $\{\Psi(\boldsymbol{\xi})\}$ are orthogonal with respect to the joint probability density function $W_{\boldsymbol{\xi}}$ of $\boldsymbol{\xi}$ and have the following properties.

$$\langle \Psi_0 \rangle = 1, \qquad \langle \Psi_i \rangle = 0, \;\; i > 0 \qquad \langle \Psi_i \Psi_j \rangle = \delta_{ij}$$

where $\langle \cdot \rangle$ is the expectation operator and $\delta_{ij}$ is the Kronecker delta. Plug this representation into the original equation.

$$\nabla \cdot \left( \alpha(\mathbf{x}, \boldsymbol{\xi}) \nabla \left( \sum_{i=0}^{N} u_i(\mathbf{x}) \Psi_i(\boldsymbol{\xi}) \right) \right) = f$$

Next we perform a Galerkin projection of the equation onto each basis polynomial by multiplying both sides by $\Psi_j$ and taking the expectation. The orthogonality leaves

$$\sum_{i=0}^{N} \nabla \cdot \left( \langle \Psi_j \alpha \Psi_i \rangle \nabla u_i \right) = \langle \Psi_j f \rangle \qquad j = 0, \ldots, N$$

with boundary conditions $u_i = 0 \in \partial\mathcal{D}$ for $i = 0, \dots, N$. Thus we are left with a system of coupled PDEs for the coefficients $\{u_i\}$ of the truncated expansion. Finally use any suitable spatial discretization – finite element, finite difference, pseudospectral – to discretize $\{u_i\}$ and solve the resulting system for the unknown values. If the spatial discretization has $P$ unknowns, then the full linear system is size $P(N+1) \times P(N+1)$. Once the coefficients are known, the moments are computed with the following formulas.

$$\mathrm{E}[u](x) = u_0(x) \qquad \mathrm{Var}[u](x) = \sum_{i=1}^{N} u_i^2 \langle \Psi_i^2 \rangle$$

The advantage to this method is that it is known to converge exponentially as $N$ increases, and it gives highly accurate results. However, by changing the governing equations, one typically has to alter any existing codes. Thus this is known as an *intrusive* method. Also, the size of the problem increases exponentially with increases in either the length of $\boldsymbol{\xi}$ or the highest degree of interpolating polynomial. If the size of the spatial discretization of the coefficients is large, this can be disastrous. The resulting linear system may or may not have special properties such as sparsity that can be exploited.

**2.3. Stochastic Collocation.** The main idea behind stochastic collocation is the recognition that moments of a random process are simply multidimensional integrals in the probability space. Therefore we can use any available deterministic quadrature rule to approximate these integrals. We describe this high-dimensional integration as given in [6]. Suppose $f$ is a one-dimensional function supported on some domain $\Gamma$ and $Q^i(f)$ is a one-dimensional quadrature rule

$$Q^i(f) = \sum_{j=1}^{m_i} w_j^i f(x_j^i), \qquad w_j^i \in \mathbb{R},\ x_j^i \in \Gamma$$

that approximates the integral

$$I(f) = \int_\Gamma f(x)\theta(x)\,dx$$

where $\theta(x)$ is a given weight function. From $Q$, we can build an $n$-dimensional rule as follows. Suppose $F(\mathbf{x})$ is a map from $\Gamma^n$ to $\mathbb{R}$ where

$$\Gamma^n = \Gamma_1 \times \cdots \times \Gamma_n$$

is a tensor product of one-dimensional domains. Then the integral

$$\tilde{I}(F) = \int_{\Gamma^n} F(\mathbf{x})\Theta(\mathbf{x})\,d\mathbf{x},$$

where $\Theta = \theta_1 \times \cdots \times \theta_n$ is the weight function, is approximated by the tensor product rule

$$\tilde{Q}^{\mathbf{i}}(F) = (Q_1^{i_1} \otimes \cdots \otimes Q_n^{i_n})(F) = \sum_{j_1=1}^{m_{i_1}} \cdots \sum_{j_n=1}^{m_{i_n}} F(x_{j_1}^{i_1}, \dots, x_{j_n}^{i_n}) \cdot (w_{j_1}^{i_1} \times \cdots \times w_{j_n}^{i_n})$$

where $\mathbf{i} = (i_1, \dots, i_n)$ is a multi-index. The problem with this method is that it requires $m_{i_1} \times \cdots \times m_{i_n}$ function evaluations to compute the integral, which can grow quite large for even a moderate number of dimensions. Instead, we can utilize a sparse grid – developed first by Smolyak [9] – to greatly reduce the number of function evaluations. The sparse grid construction is given by the formula

$$A(q,n)(F) = \sum_{q-n+1\le|\mathbf{i}|\le q} (-1)^{q-|\mathbf{i}|} \cdot \binom{d-1}{q-|\mathbf{i}|} \cdot (Q_1^{i_1} \otimes \cdots \otimes Q_n^{i_n})(F)$$

where $q$ is a parameter that controls the accuracy. This formula takes a linear combination of intelligently chosen tensor product grids to maximize a certain metric for accuracy. These tensor grids overlap at many points, thus reducing the total number of points required to evaluate the rule.

If the $n$-dimensional rule is built from $Q^i$'s that are nested, i.e. the set of points used to evaluated $Q^i$ is a subset of the points used to evaluate $Q^{i+1}$, then the sequence of $n$-dimensional rules are also nested. For example, this happens when using one-dimensional Clenshaw-Curtis and Gauss-Patterson rules. In the end, each of these $n$-dimensional formulas can be written as a linear combination of function evaluations and weights.

To compute the moments of the solution to the model problem, we employ the quadrature rules.

$$\begin{aligned} \mathrm{E}[u](\mathbf{x}) &= \int_\Omega u(\mathbf{x},\boldsymbol{\xi}) W_{\boldsymbol{\xi}}(\boldsymbol{\xi})\, d\boldsymbol{\xi} \\ &\approx \sum_{i=1}^{N} u(\mathbf{x},\boldsymbol{\xi}_i) W_{\boldsymbol{\xi}}(\boldsymbol{\xi}_i) w_i \\ &\equiv \mu_u \end{aligned}$$

$$\begin{aligned} \mathrm{Var}[u](\mathbf{x}) &= \int_\Omega (u(\mathbf{x},\boldsymbol{\xi}) - \mathrm{E}[u(\mathbf{x},\boldsymbol{\xi})])^2 W_{\boldsymbol{\xi}}(\boldsymbol{\xi})\, d\boldsymbol{\xi} \\ &\approx \int_\Omega (u(\mathbf{x},\boldsymbol{\xi}) - \mu_u)^2 W_{\boldsymbol{\xi}}(\boldsymbol{\xi})\, d\boldsymbol{\xi} \\ &\approx \sum_{i=1}^{N} (u(\mathbf{x},\boldsymbol{\xi}_i) - \mu_u)^2 W_{\boldsymbol{\xi}}(\boldsymbol{\xi}_i) w_i \end{aligned}$$

Each $u(\mathbf{x},\boldsymbol{\xi}_i)$ is a solution to a deterministic problem

$$\nabla \cdot (\alpha(\mathbf{x},\boldsymbol{\xi}_i)\nabla u_i(\mathbf{x},\boldsymbol{\xi}_i)) = f \qquad \mathbf{x} \in \mathcal{D}.$$

In practical terms, we run $N$ deterministic solves of an existing code and post-process the results to compute the moments.

Stochastic collocation [5] combines the non-intrusive features of Monte Carlo with the spectral convergence rate of stochastic Galerkin. Also, if the length of $\boldsymbol{\xi}$ is sufficiently small, the stochastic collocation typically requires much fewer deterministic solves than Monte Carlo to achieve some desired accuracy. However, for high dimensional integrals, i.e. the length of $\boldsymbol{\xi}$ greater than 10 or so, the number of points, and therefore deterministic solves, can still get relatively large depending on the computational budget.

**3. Method of Manufactured Solutions.** The Method of Manufactured Solutions (MMS) is a procedure used for verifying the order-of-accuracy of a computer code which solves partial differential equations [8]. By successfully verifying the order-of-accuracy it is claimed that the code is free of all order-impacting coding mistakes. The method has also proven valuable for identifying algorithmic weaknesses and deficiencies. The method is particularly valuable in cases where an exact analytic solution cannot be derived. Systematic mesh refinement is a key element of order-verification.

The MMS procedure consists of the following steps

(1) Determine the governing set of equations solved by the code, and the formal order-of-accuracy of the solution method, if known.
(2) Construct an exact solution to the equations,
(3) Run the code using input that is expected to generate the corresponding correct numerical solution,
(4) Calculate the global discretization error,
(5) Refine the grid and repeat steps 3 and 4 until the numerical solution appears to converge,
(6) Calculate the observed order-of-accuracy from the set of numerical solutions, and compare it to the formal or expected order-of-accuracy.

If the formal and observed order-of-accuracy agree (to within some tolerance), then the code is said to have 'passed' the test constituting the MMS procedure.

To construct the manufactured solution, a function $\tilde{u}$ is selected and substituted into the PDE. The result of this substitution is known as the 'source term.' The source term is then input to the code (or internally coded) so that the code can reproduce the manufactured solution. As a simple example of constructing a manufactured solution, suppose one wanted to verify the order-of-accuracy of a code solving the 1D non-linear equation

$$u_{xx} + \cos(\pi u) = 0$$

on the interval $[-1, 1]$ with boundary conditions

$$\begin{aligned} u(-1) &= u_0 \\ (ku)_x(1) &= h(x) \end{aligned}$$

Since the equations have no obvious analytic solution, we manufacture one by choosing $\tilde{u} = \sin(\pi x)$ and $k(x) = \sqrt{1+x}$. Then, $\tilde{u}$ satisfies

$$u_{xx} + \cos(\pi u) = -\pi^2 \sin(\pi x) + \cos(\pi \sin(\pi x))$$

The right-hand-side is thus the input source term needed to balance the interior equation. Similarly, the input $u_0 = \tilde{u}(-1) = 0$ and the input

$$h(x) = \pi\sqrt{1+x}\cos(\pi x) + \frac{\sin(\pi x)}{2\sqrt{1+x}}$$

With these inputs to the code, the code is run with a set of successively refined meshes and the discretization error computed. Details of the MMS procedure are given in [3].

**4. Stochastic MMS.** The basic idea behind stochastic MMS is the same as in the deterministic case. This method can verify the codes used to compute the moments for each of the techniques mentioned above – both intrusive and non-intrusive. In the following section, we describe the method in general terms and then give three specific examples.

**4.1. General Case.** Consider the general differential equation with stochastic inputs

$$\mathcal{L}(\mathbf{x}, t, \boldsymbol{\xi}; u) = 0, \qquad \mathbf{x} \in \mathcal{D} \tag{4.1}$$

$$u|_{\partial\mathcal{D}} = g(\mathbf{x}, t, \boldsymbol{\xi}) \tag{4.2}$$

$$u(\mathbf{x}, 0, \boldsymbol{\xi}) = u_0 \tag{4.3}$$

where the dependence on $\boldsymbol{\xi}$ implies that the operator may involve some finite number of parameterizing random variables. Here $u = u(\mathbf{x}, t, \boldsymbol{\xi})$ is a function that satisfies the equations above, and $f(\mathbf{x}, t, \boldsymbol{\xi})$ is the source term. The operator $\mathcal{L}$ can be linear or non-linear, and can depend on space and time.

Suppose we have a code that computes the moments of the solution to 4.1 that we want to use to perform an order verification study. The only requirement on the code is that it is able to accept an arbitrary forcing term – or right hand side. For the intrusive methods of computing moments, this means the right hand side (and its projection onto each Galerkin basis) must be specified in something like a user-defined function. For the non-intrusive methods, the existing deterministic code must accept user-defined forcing terms.

Choose a function $\bar{u}(\mathbf{x}, t, \boldsymbol{\xi})$ that satisfies the boundary and initial conditions. (Note that this requirement is *not* necessary in the deterministic case.) This $\bar{u}$ should be an explicit function of the components of $\boldsymbol{\xi}$. Next compute the moments of $\bar{u}$, $\mathrm{E}[\bar{u}]$ and $\mathrm{Var}[\bar{u}]$ using the definitions of expectation and variance. This can often be done analytically; if necessary, it can be computed with numerical integration. With these "exact" moments, we can check the convergence of the given code.

To check the convergence rate of the code, first compute

$$\bar{f}(\mathbf{x}, t, \boldsymbol{\xi}) = \mathcal{L}(\mathbf{x}, t, \boldsymbol{\xi}; \bar{u})$$

offline either by hand or with the aid or a computer algebra system. Next create a user-defined function in the code that represents $\bar{f}$. For an intrusive stochastic Galerkin code, this may involve computing the inner product of $\bar{f}$ against the basis polynomials. For non-intrusive codes, it will involve evaluating $\bar{f}$ at each sample point or collocation point and running separate solves. Thus the user-defined right hand side will need to take into account the varying values of the random input parameters.

For an intrusive stochastic Galerkin code, choose an order of the truncated polynomial representation of the solution, and use the code to compute the moments $\mathrm{E}[u]$ and $\mathrm{Var}[u]$ with $\bar{f}$ as the forcing term. By increasing the order of the truncated expansion and comparing $\mathrm{E}[u]$ and $\mathrm{Var}[u]$ with the exact $\mathrm{E}[\bar{u}]$ and $\mathrm{Var}[\bar{u}]$, respectively, we can verify the convergence rate of the code.

For a non-intrusive code, choose a level of accuracy of the quadrature rule – which sets the number of points in the rule – and, again, use the code to compute the moments $\mathrm{E}[u]$ and $\mathrm{Var}[u]$ with $\bar{f}$ as the forcing term. Verify the convergence rate of the code by increasing the level of the quadrature rule and comparing the computed moments with the exact $\mathrm{E}[\bar{u}]$ and $\mathrm{Var}[\bar{u}]$.

The metric we use to verify the rate of convergence is the infinity norm of the difference between the compute and exact moments, e.g.

$$||\mathrm{Var}_N[u] - \mathrm{Var}[\bar{u}]||_\infty,$$

where the subscript $N$ denotes the appropriate dependence on discretization in the random space. In the next sections, we give three concrete examples of using this method to verify codes.

**4.2. Example 1.** Consider the following 1-D elliptic equation.

$$-\xi u_{xx} = f \qquad x \in [0, 2\pi]$$

where $\xi$ is a uniformly distributed random variable over the interval $[a, b]$ with $0 < a < b$, and the boundary conditions are $u(0, \xi) = u(2\pi, \xi) = 0$.

We can choose $\bar{u} = \xi \sin(x)$ since it is an explicit function of $\xi$ and

$$\xi \sin(0) \;=\; \xi \sin(2\pi) \;=\; 0.$$

The mean and variance of $\bar{u}$ are given by

$$\begin{aligned}
\mathrm{E}[\bar{u}] &= \mathrm{E}[\xi \sin(x)] \\
&= \mathrm{E}[\xi] \sin(x) \\
&= \frac{b+a}{2} \sin(x)
\end{aligned}$$

$$\begin{aligned}
\mathrm{Var}[\bar{u}] &= \mathrm{E}[\bar{u}^2] - (\mathrm{E}[\bar{u}])^2 \\
&= \mathrm{E}[\xi^2] \sin^2(x) - \left(\frac{b+a}{2} \sin(x)\right)^2 \\
&= \left(\frac{b^2+ab+a^2}{3} - \left(\frac{b+a}{2}\right)^2\right) \sin^2(x) \\
&= \frac{(b-a)^2}{12} \sin^2(x)
\end{aligned}$$

Next compute

$$\begin{aligned}
\bar{f} &= \xi \bar{u}_{xx} \\
&= -\xi^2 \sin(x)
\end{aligned}$$

Plug $\bar{f}$ into the given code for computing moments.

For an intrusive stochastic Galerkin code, choose the truncation level $P = P_{\text{init}}$ of the series

$$u \approx \sum_{i=0}^{P} u_i(x) \Psi_i(\xi)$$

where $\{\Psi_i\}$ are the Legendre polynomials which are orthogonal with respect to the uniform measure. Run the given code varying $P$ from $P_{\text{init}}$ to $P_{\text{final}}$ to compute values for E[$u$] and Var[$u$]. Compare these computed values to E[$\bar{u}$] and Var[$\bar{u}$] and observe the convergence as $P$ increases. Using a short Matlab implementation of a stochastic Galerkin approach to this problem, we get the results in Figure 4.2. We can see the error drop to machine precision after one term in the PCE. This makes sense when we compare the manufactured solution to the PCE representation. Observe that

$$\xi \sin(x) = \sum_{i=0}^{N} u_i(x) \Psi_i(\xi)$$

implies that only the first two coefficients in the expansion are non-zero. (Note that there is a change in variables from $\xi$ to a uniform[-1,1] random variable to recover the Legendre polynomials.)

**4.3. Example 2.** In this example we incorporate another parameterizing random variable into the equation. This problem can easily be extended to an arbitrary number of parameterizing random variables, but we stick to two for the sake of demonstration. Consider

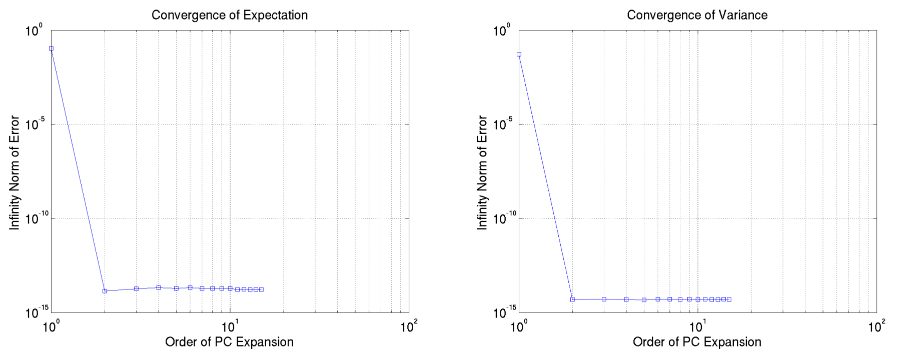


FIG. 4.1. *Convergence of computing mean and variance with stochastic Galerkin on a 1-D elliptic problem with a random coefficient.*

the model elliptic problem with one spatial dimension, $x \in [0, 2\pi]$, and the spatially varying coefficient given by

$$\alpha(x, \xi_1, \xi_2) = \xi_1 \cos(\omega_1 x) + \xi_2 \cos(\omega_2 x) + 2$$

where $\xi_1$ and $\xi_2$ are independent and distributed uniformly over the interval $[-1, 1]$. The wave numbers are chosen as $\omega_1 = 1$ and $\omega_2 = 2$.

For this problem, we choose $\bar{u} = \xi_1\xi_2 \sin(x)$ and compute the exact moments as

$$\begin{aligned} \mathrm{E}[\bar{u}] &= \mathrm{E}[\xi_1\xi_2 \sin(x)] \\ &= \mathrm{E}[\xi_1]\mathrm{E}[\xi_2] \sin(x) \quad \text{by independence} \\ &= 0 \end{aligned}$$

$$\begin{aligned} \mathrm{Var}[\bar{u}] &= \mathrm{E}[\xi_1^2\xi_2^2 \sin^2(x)] \\ &= \mathrm{E}[\xi_1^2]\mathrm{E}[\xi_2^2] \sin^2(x) \quad \text{by independence} \\ &= \frac{1}{9} \sin^2(x) \end{aligned}$$

To compute the manufactured right hand side, we derive

$$\begin{aligned} \bar{f} &= -\nabla \cdot (\alpha \nabla \bar{u}) \\ &= -((\xi_1 \cos(\omega_1 x) + \xi_2 \cos(\omega_2 x) + 2)(\xi_1\xi_2 \sin(x))_x)_x \\ &= (\xi_1\omega_1 \sin(\omega_1 x) + \xi_2\omega_2 \sin(\omega_2 x))\xi_1\xi_2 \cos(x) + (\xi_1 \cos(\omega_1 x) + \xi_2 \cos(\omega_2 x) + 2)\xi_1\xi_2 \sin(x) \end{aligned}$$

For this example, we verify the $\sqrt{M}$ convergence rate of a Monte Carlo method for computing the moments. We ran this study with Sandia's DAKOTA Toolkit [1] and a short code for solving a deterministic problem where $\xi_1$ and $\xi_2$ are input parameters. The plots of convergence are given in Figure 4.3.

**4.4. Example 3.** For the final example, we consider a time-dependent Burgers' equation with one spatial dimension. We can state the problem as: compute the moments of $u(x, t, \xi)$ where $u$ satisfies

$$u_t + \left(\frac{u^2}{2}\right)_x = 0$$

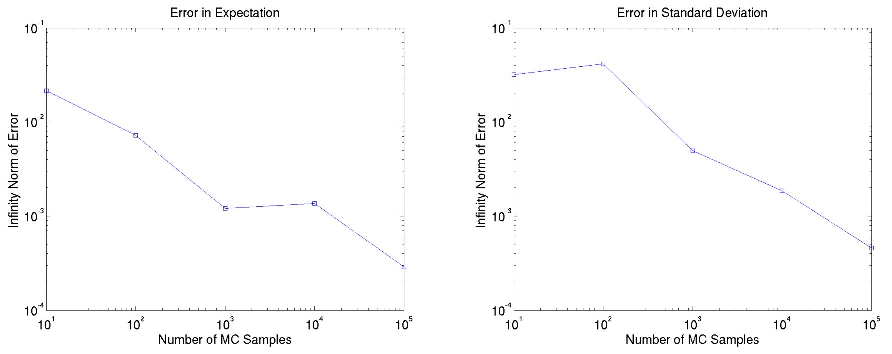


FIG. 4.2. *Convergence of computing mean and std. dev. with Monte Carlo on an elliptic problem with random coefficients.*

for $x \in [0, 1]$, $t \in [0, T]$, and almost every value of $\xi$. Instead of specifying the boundary and initial conditions, we will derive them from the manufactured solution. We choose

$$\bar{u} = \sin(2\pi x + t) + \sigma\xi^{10}$$

where $\xi$ is uniformly distributed over $[-1, 1]$ and $\sigma$ is a parameter that controls the total variation. From this we derive the initial and boundary conditions of the problem:

$$u(x, 0, \xi) = \sin(2\pi x) + \sigma\xi^{10} \quad u(0, t, \xi) = u(1, t, \xi) = \sin(t) + \sigma\xi^{10}$$

Note that the manufactured solution is a tenth degree polynomial in terms of $\xi$. Let us now compute the exact moments. It is useful to recall the following formula for $\xi$:

$$\mathrm{E}[\xi^n] = \frac{1 - (-1)^{n+1}}{2(n + 1)}.$$

Then we can compute

$$\begin{aligned} \mathrm{E}[\bar{u}] &= \mathrm{E}[\sin(2\pi x + t) + \sigma\xi^{10}] \\ &= \sin(2\pi x + t) + \sigma\mathrm{E}[\xi^{10}] \\ &= \sin(2\pi x + t) + \frac{\sigma}{11} \end{aligned}$$

$$\begin{aligned} \mathrm{E}[\bar{u}^2] &= \mathrm{E}[(\sin(2\pi x + t) + \sigma\xi^{10})^2] \\ &= \mathrm{E}[\sin^2(2\pi x + t) + 2\sigma\xi^{10}\sin(2\pi x + t) + \sigma^2\xi^{10}] \\ &= \sin^2(2\pi x + t) + \frac{2\sigma}{11}\sin(2\pi x + t) + \frac{\sigma^2}{21} \end{aligned}$$

so that

$$\mathrm{Var}[\bar{u}] \;=\; \mathrm{E}[\bar{u}^2] - (\mathrm{E}[\bar{u}])^2 \;=\; \frac{100\sigma^2}{2541}.$$

Using a 1-D Gaussian quadrature rule as above to approximate the moments, we obtain the plots in Figure 4.4 for convergence to the exact moments. This convergence is admittedly not as clean as the first two examples. In this case the error is dominated by the spatial discretization. We used a first order method to solve the deterministic Burgers' equation. In future work we will discuss the relationship between the error associated with the spatial discretization and the discretization in the random space.

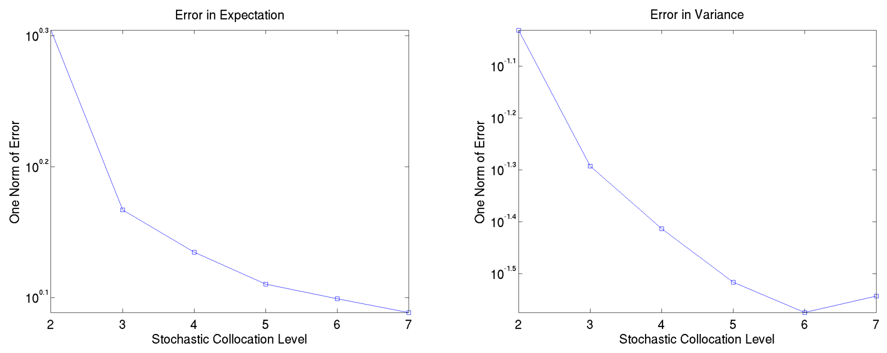


FIG. 4.3. *Convergence of moments with stochastic collocation for Burgers' equation*

**5. Summary.** With the development of uncertainty quantification algorithms for computing moments of solutions of PDEs with random inputs, a need has arisen for verification of the implementations of these algorithms in the sense of both correctness and rate of convergence. In this paper, we proposed a stochastic method of manufactured solutions to satisfy the need. Simply apply the random operator of the PDE to a manufactured function of the appropriate variables to derive a right hand side. This results in a related problem with an analytical (manufactured) solution with which to verify the correctness and rate of convergence of the implementation.

At the very least, this work provides a method to create benchmark problems that test the robustness of new UQ methods and provides a means to compare existing methods.

## REFERENCES

[1] M. S. Eldred, A. A. Giunta, B. G. van Bloemen Waanders, S. F. Wojtkiewicz, W. E. Hart, and M. P. Alleva, *DAKOTA, a multilevel parallel object-oriented framework for design optimization, parameter estimation, uncertainty quantification, and sensitivity analysis*, Technical Report SAND2001-3796, Sandia, (2002).

[2] R. Ghanem and P. Spanos, *Stochastic Finite Elements: A Spectral Approach*, Springer-Verlag, New York, 1991.

[3] P. Knupp and K. Salari, *Verification of Computer Codes in Computational Science and Engineering*, Chapman and Hall/CRC, 2003.

[4] M. Loeve, *Probability Theory*, Van Nostrand Company, Inc., New York, N. Y., 1955.

[5] F. Nobile, R. Tempone, and C. G. Webster, *A sparse grid stochastic collocation method for elliptic partial differential equations with random input data*, Technical Report #85, MOX, Dipartimento di Matematica, (2006).

[6] E. Novak and K. Ritter, *High dimensional integration of smooth functions over cubes*, Numer. Math., 75 (1996), pp. 79–97.

[7] J. A. Rice, *Mathematical Statistics and Data Analysis*, Duxbury Press, 2nd ed., 1995.

[8] P. Roache, *Code verification by the method of manufactured solutions*, Journal of Fluids Engineering, Transactions of the ASME, 124 (2002), pp. 4–10.

[9] S. Smolyak, *Quadrature and interpolation formulas for tensor products of certain classes of functions*, Dokl. Akad. Nauk SSSR 4, (1963), pp. 240–243.

[10] N. Wiener, *The homogeneous chaos*, Amer. J. Math., 60 (1938), pp. 897–936.

[11] D. Xiu and G. Karniadakis, *The Wiener-Askey polynomial chaos for stochastic differential equations*, SIAM Journal on Scientific Computing, 24 (2002), pp. 619–644.

## Architecture and Systems Software

All applications software lies on top of systems software, which in turn manages and controls the underlying computer hardware. As such, the development of improved systems software and the development of improved computer architectures enables improved performance for all applications. The articles in this section discuss systems software and architectural improvements to enhance performance for real-world scientific applications.

Wheeler and Murphy introduce the qthread API and a Unix implementation. This abstraction provides basic lightweight thread control and synchronization primitives portable to existing highly parallel architectures, enabling lighter-weight threading on common operating systems. Rupnow and Underwood present a RFU architecture to efficiently handle a large number of input and output operands, and demonstrate speedup on many computational kernels for real scientific applications. La Fratta *et al.* explore an alternative execution model in which groups of dependent floating point operations are encoded into a single instruction, called a floating point instruction aggregate (FPIA). They present the requirements of an architecture to support such instructions and estimate potential performance improvements offered by such an architecture. Lastly, Curry *et al.* explore a GPU-based system for generating redundant information for error recovery in a RAID system that can correct for the simultaneous loss of two or more disks within a single array.

M.L. Parks
S.S. Collis

December 6, 2007

# LIGHTWEIGHT THREADING FOR ARCHITECTURAL DESIGN RESEARCH

KYLE B. WHEELER* AND RICHARD C. MURPHY†

**Abstract.** Increased parallelism is a powerful method for increasing computational power, and one way that such parallelism is expressed is as threads. Large scale threading benefits significantly from lower per-thread overhead, and lightweight threading with hardware support is a developing area of research. Each architecture with support for lightweight threading provides a different interface to the programmer, making comparisons between them difficult. As such there is a need for an abstraction that provides basic lightweight thread control and synchronization primitives in a way that is portable to existing highly parallel architectures as well as to future and potential architectures to assist in exploring both the architectural needs of large scale threading and the extent to which threading can be expressed in existing programming languages. This paper introduces the qthread API and its Unix implementation.

**1. Introduction.** Modern supercomputers have largely taken the route of parallel computation to achieve increased computational power. Use of parallelism to increase execution speed has largely operated in two entirely different worlds: the world of the very large, where programmers explicitly create parallel threads of execution, and the world of the very small, where processors attempt to extract parallelism from streams of instructions. Parallelism has been exploited at low levels of the architecture, including hardware-based techniques such as Simultaneous Multi-Threading and out-of-order execution, both of which identify work that is sufficiently unrelated to be performed at the same time. However, programmers have only been given interfaces such as MPI [5], Pthreads [8], and more recently OpenMP [4] to express the parallelism of their algorithms. This is a significant semantic disconnect that limits or prevents full exploitation of the available parallelism.

As the amount of available lower-level hardware parallelism increases, it becomes more and more necessary to allow the programmer to express parallel execution simply, and without the overhead usually associated with standard parallel programming models.

Support for lightweight threading in hardware is a rapidly developing field, and current portable programming interfaces do not expose these newly available capabilities to the programmer for use on architectures that have them.

The qthread API is designed to change that. Designed to be sufficiently similar to existing threading libraries so as to be simple to use, the qthread API does not have many features that heavyweight threading uses its large amount of thread state to provide. Qthreads, by contrast, have very small amounts of thread-specific state, and provide initialization-free synchronization methods based around the Full/Empty Bit (FEB) synchronization technique [10] that interact directly with their simple scheduling mechanism. While being lightweight, the qthread API is designed to be simple to port to alternative and experimental architectures, such as Cray's Multi-Threaded Architecture (MTA) [1] and the developing Processor-In-Memory (PIM) [9, 3, 6] designs.

**2. Design Goals.** The qthread API was designed to maximize portability to experimental architectures supporting lightweight threads and unusual synchronization primitives while providing a stable interface to the programmer for using these lightweight threads. Lightweight threads, and thus qthreads, have inherent restrictions on their stack size, and provide an easily emulated universal locking scheme based on the Full/Empty Bit (FEB) concept [10]. For control of resources, qthreads can use a definition of a thread, called a "future", that is only created when there are resources available for it.

In addition to portability, performance is important to the design of the API. The API's goal is to add as little overhead as possible to experimental architectural threading primitives

*University of Notre Dame, kwheeler@cse.nd.edu
†Sandia National Laboratories, rcmurphy@sandia.gov

while maintaining its generic nature. For example, to reduce overhead, qthreads are considered mostly anonymous, and cannot be affected directly by other threads—there is no way to cancel a currently running thread, and threads are not expected to respond to signals. The sole way to communicate reliably between threads is to use shared data structures protected by the locking functions provided by the API. Waiting for threads to finish, for example, is best done by waiting for them to return a value, which allows all qthreads to release their resources when they exit even if no one is waiting for them.

The generic design of the API is the primary contribution of this work, though the Unix implementation of the API is important to demonstrate the workability and functionality of the system. The development of the implementation informed the development of the API design. For example, since lightweight threads have limited stack space, a function was needed in the API that would allow the programmer to monitor this limited resource, for debugging and runtime decision-making. Both the API and the implementation, with the exception of the futurelib, were designed and implemented entirely by the authors of this paper. The futurelib component was designed and implemented by Megan Vance.

**2.1. Basic Design.** The qthread API consists of four components: the core lightweight thread interaction command set, a set of commands for resource-limit-aware threads ("futures"), a C++ interface for basic threaded loops, and a C-language interface for several varieties of basic threaded loops. The loop packages are built on top of the core thread control and resource-limit-aware threading components.

**2.2. Basic Thread Control.** The qthread library implements a nearly-anonymous threading interface. Threads, once created, cannot be controlled by other threads. However, they can provide return values which are protected by a synchronization method, such as an FEB, enabling a thread to wait for another thread to complete. When threads are waiting for a synchronization event—i.e. "blocked"—they are removed from the scheduling queues and are only available to be scheduled when they are unblocked.

Threads must assume that they are running in a cooperatively scheduled environment (although they may not be), and thus spin-locking rather than using one of the provided synchronization primitives is discouraged.

Lightweight threads are scheduled in the bailiwick of one of several "shepherds." The number of shepherds is defined when the library is initialized. A shepherd is a grouping construct, or a thread mobility domain. Qthreads are assigned a shepherd when they are created and cannot move between shepherds. In the Unix version of the qthread library, a shepherd is a pthread responsible for organizing and executing the qthreads. In other qthread implementations, shepherds may be nodes in the system, memory regions, or protection domains.

The outline of the qthread thread-manipulation API is given below. For a complete specification, see the qthread library's documentation:

qthread_init(n) Initialize the threading library, able to use n shepherds.

qthread_fork(func,arg,ret) Create a thread to run func(self,arg) and make it available for execution. Its return value will be stored in ∗ret. The thread may be created on any shepherd.

qthread_fork_to(func,arg,ret,shep) Create a thread and make it available for execution on the shepherd shep.

qthread_self() returns (self). Obtain a reference to the executing thread. This is primarily for use in avoiding extra lookups when doing locking; use of this reference by other threads is undefined.

qthread_shep() returns (shepherd). Discover which shepherd this thread has been assigned to.

qthread_retloc() returns (addr). Discover where the return value for this function will be stored.

qthread_stackleft() returns (bytes). Discover approximately how many bytes are available in the thread's stack.

qthread_finalize() Destroys all threads, releases all related memory.

The qthread API also provides functions to manipulate locking structures. These mimic both a generic mutex and full/empty bits, though the implementation of each may not have direct hardware support. The two categories of synchronization may be implemented using the same underlying mechanism, or may not, so using the two techniques on the same addresses at the same time results in undefined behavior. When these locking primitives do not have direct hardware support, they must be emulated using what locking primitives are available. The current Unix implementation implements them using pthread mutexes, and a 32-way striped mutex-protected hash table.

The outline of the qthread locking API is given below. For a complete specification, see the qthread library's documentation:

qthread_lock(self,addr) Obtain a lock on the address addr.

qthread_unlock(self,addr) Release the lock on the address addr.

qthread_empty(self,addr) Set the full/empty state of address addr to be empty.

qthread_fill(self,addr) Set the full/empty state of address addr to be full.

qthread_readFE(self,dest,src) Wait for src to become full, then copy the data in ∗src into ∗dest and set src to be empty. This is atomic.

qthread_readFF(self,dest,src) Wait for src to become full and then copy the data in ∗src into ∗dest. This is atomic.

qthread_writeEF(self,dest,src) Wait for dest to become empty and then copy the data in ∗src into ∗dest and set dest to be full. This is atomic.

qthread_writeF(self,dest,src) Copy the data in ∗src into ∗dest and set dest to be full. This is atomic.

qthread_feb_status(addr) returns (status). Discover the current full/empty state of addr.

qthread_incr(addr,incr) returns (value). Discover the current value stored at addr and add incr to it. May or may not use qthread_lock() and qthread_unlock(), depending on compile-time options and hardware/compiler support. This is atomic.

**2.3. Futures.** The qthread API has no built-in limitations on the number of threads spawned other than the amount of memory necessary for the threads' contexts. Memory allocation failures are the only limit. When confronted with the memory limits, applications have few options other than to wait and try again. To make memory management easier, a variant of qthread creation was created to allow the programmer to set arbitrary limits on the number of threads that may exist at the same time. These resource-limit-aware threads are called "futures". Attempting to create a future when the maximum number of futures already exist causes the creating thread to block until the future can be successfully created. The future-related API is a simple extension to the qthread thread-manipulation API:

future_init(limit) Defines the maximum number of futures per shepherd to be limit.

future_fork(func,arg,ret) Essentially identical to qthread_fork(), but creates a future and as such will block until the future has been created.

future_yield(self) Causes a future to become merely a qthread temporarily. Useful for avoiding deadlock in some situations.

future_acquire(self) Causes a qthread to become a future if it had previously yielded; may block if there are too many futures already.

The qthread API also includes some convenience functions, built on top of the core threading, locking, and futures functionality. These convenience functions generally provide threaded loops and the ability to perform simple actions over large sets of data. These convenience functions are separated into three categories: the futurelib (written by Megan Vance), the Qloops, and Qutils.

**2.3.1. Futurelib.** The futurelib is a template-based C++ interface to the qthread and futures primitives. The most important and basic functions in the futurelib are mt_loop(), for parallel iterations that do not return values, and mt_loop_returns(), for parallel iterations that do. The distinction between the two is not always obvious, and can often be treated as merely a programmer convenience. Both functions are implementations of a parallel for-loop. The mt_loop() function is used in a format like this:

```
mt_loop<...argtypelist..., looptype>
    (function, ...arglist..., startval, stopval, stepval);
```

This construction is relatively straightforward. The function argument specifies a function that will be used for each iteration of the loop. The iterations are defined as each having a number starting at startval, ending at stopval, incremented by stepval (which is optional, and defaults to 1). The *looptype* option specifies the kind of parallelism, and has four possible values:

mt_loop_traits ::Par All iterations are created at once as qthreads and the mt_loop() function will not return until all iterations are finished.

mt_loop_traits ::ParNoJoin Similar to Par, but does not wait for iterations to finish before returning flow control to the parent thread.

mt_loop_traits ::Future Similar to Par, but uses futures instead of qthreads.

mt_loop_traits ::FutureNoJoin Similar to ParNoJoin, but uses futures instead of qthreads.

The argtypelist in the mt_loop() construction is a list of conceptual types defining how the arguments to the iteration function will be handled. Each conceptual type corresponds to a single argument to the iteration function. Valid conceptual types are:

Iterator This corresponds to the integer value associated with each iteration, being a number between startval and stopval.

ArrayPtr This corresponds to an argument that is a pointer to an array. Each iteration function instance will be passed the value of array[ iteration ].

Ref The corresponding argument will be passed to the iteration function as a reference.

Val The corresponding argument will be passed to the iteration function as a constant value (i.e. the same value will be passed to all iterations).

As an example, here is a simple loop:

```
for (int i = 0; i < 10; i++) {
    array[i] = i;
}
```

This simple loop could be threaded with the futurelib like so:

```
void assign(int &array_value, const int i) {
    array_value = i;
}

mt_loop<ArrayPtr, Iterator, mt_loop_traits::Par>
       (assign, array, 0, 0, 10);
```

Alternatively, the following code would achieve the same goal:

```
void assign(const int i, const int * a) {
    a[i] = i;
}
```

```
mt_loop<Iterator, Val, mt_loop_traits::Par>
       (assign, 0, array, 0, 10);
```

The mt_loop_returns() variant adds the specification of what to do with the return values. The pattern is like this:

```
mt_loop_returns<returnvaltype, ...argtypelist..., looptype>
    (retval, function, ...arglist..., startval, stopval, stepval);
```

The only difference is in the *returnvaltype* and the retval. The *returnvaltype* can be either an ArrayPtr or a Collect. If it is an ArrayPtr, each return value will be stored in a separate entry in the retval array, corresponding to the iteration. The parallel loop will behave similar to the following loop:

```
for (int i = startval; i < stopval; i += stepval) {
    retval[i] = function(...arglist...);
}
```

The Collect type is more interesting, and can be one of the following:

Collect<mt_loop_traits::Add> This sums all of the return values in parallel and stores the value in retval.

Collect<mt_loop_traits::Sub> This subtracts all of the return values in parallel and stores the value in retval. Note that the answer may be nondeterministic.

Collect<mt_loop_traits::Mult> This multiplies all of the return values in parallel and stores the value in retval.

Collect<mt_loop_traits::Div> This divides all of the return values in parallel and stores the value in retval. Note that the answer may be nondeterministic.

The accumulation functions occur as data is made available by iteration functions returning. Thus, adding or multiplying multiple integers is predictable, while doing the same with floating point numbers exposes the user to potential rounding problems due to the operation ordering. Use of Collect<mt_loop_traits::Add> is roughly equivalent to the following loop:

```
for (int i = startval; i < stopval; i += stepval) {
    retval += function(...arglist...);
}
```

**2.3.2. Qloop, and Qutil.** The qloop component provides futurelib-like functionality with a strictly C-language interface. Because of the inherent limitations of the C language, this interface is slightly more cumbersome than the futurelib's C++ interface. The qloop component provides two alternative parallel loop behaviors: one that, like futurelib, spawns a separate qthread or future for each iteration, and the other which spawns one qthread for each shepherd and gives each thread a segment of the iteration-space to compute.

An outline of the qloop parallel for-loop interface is given below. For the complete specification, see the qthread documentation:

qt_loop(start,stop,stride,func,argptr) Spawns func as a qthread with the argument argptr for every iteration between start and stop with a stride of stride.

qt_loop_future(start,stop,stride,func,argptr) Similar to qt_loop(), but uses futures instead of qthreads.

qt_loop_balance(start,stop,func,argptr) Spawns func as a qthread for each shepherd. Each instance of func is assigned a specific range of the iteration space between start and stop over which it can operate.

qt_loop_balance_future( start ,stop,func,argptr) Similar to qt_loop_balance(), but uses futures instead of qthreads.

qt_loopaccum_balance(start,stop,size,out,func,arg,accfunc) Similar to qt_loop_balance(), however each func may have a return value size bytes long that will be stored in out. The combination of the return values is controlled by the accumulator function accfunc.

As an example, qt_loop () behaves similarly to the following loop:

```
for (int i = start; i < stop; i += stride) {
    func(argptr);
}
```

The qt_loop_balance() function behaves like the following loop. For simplicity, this example loop assumes that the number of iterations is evenly divisible by the number of shepherds, though qt_loop_balance() does not. Note that it has access to the number of shepherds, which is stored internally in the qthreads library:

```
int stride = (stop - start)/num_shepherds;
for (int i = start; i < stop; i += stride) {
    func(i, i+stride, argptr);
}
```

Additionally, qloop provides several simple utility functions that use the balanced loop interface to perform simple tasks, such as compute the sum of every element in an array. The qutil component also provides utility functions for performing these simple tasks, but implements them with a lagging-loop structure. Table 2.1 compares the equivalent functions for operating on arrays of doubles. Similar functions are available for integers and unsigned integers.

| Qloop | Qutil |
|---|---|
| qt_double_min() | qutil_double_min () |
| qt_double_max() | qutil_double_max() |
| qt_double_prod() | qutil_double_prod () |
| qt_double_sum() | qutil_double_sum() |

TABLE 2.1
*Qloop compared to Qutil*

Assuming that each shepherd maps to a single processor's ability to execute, the balanced loop design provided by qt_loop_balance() allows the computational power of a parallel system to be maximized with a minimum amount of overhead, but presumes that the iterations do not interact or depend on each other in any way, and does not compensate for additional threads that may also be executing.

The lagging-loop design used in Qutil functions spawns threads for fixed-size segments of the iteration space. The number of threads active at any given time is thus either limited only by the problem size, or limited by the imposed limit on the number of active futures. This technique cooperates with unrelated executing threads well, and handles interaction between threads better because there are typically more than one thread assigned to each shepherd, so when one blocks, other threads can execute. However, this behavior incurs more thread-management overhead, which varies by implementation. This loop design also requires a well-chosen segment size, to minimize thread overhead without reducing the parallelism to a point where it cannot cooperate with other unrelated threads sufficiently. In that sense, a balanced loop design is a special case of the lagging-loop design, where the library chooses the segment size to match the available shepherd-level parallelism.

**3. Application Development.** Development of software that realistically takes advantage of lightweight threading is important to research, but difficult to achieve, as there are few organizations willing to devote the time to develop large scale threaded applications for architectures that may never exist.

**3.1. Cray's Multi-Threaded Architecture.** Cray's MTA [1] is of particular interest to lightweight threading researchers, as it provides lightweight threads, fast locks, FEB support, and a toolchain to take advantage of them. The architecture even has some large-scale software available for it. In particular, the Multi-Threaded Graph Library (MTGL) [2] provides a good example of high-performance code written for an architecture that provides these capabilities.

The platform-specific features of the MTA are primarily accessed through the use of C-language pragmas that instruct the custom MTA compiler how to parallelize the code. Because MTA applications are so dependent on the compiler, porting such code to other architectures is rather difficult.

Because the qthread library provides similar features, the MTGL can be ported relatively easily to use the qthread API rather than the native MTA API, and once that is done, can be run on other architectures relatively easily, including any experimental architectures for which a qthread implementation has been developed.

**3.2. High Performance Computing Conjugate Gradient Benchmark.** The qthread API makes parallelizing ordinary serial code extremely simple. As a demonstration of its capabilities, the HPCCG benchmark written by Mike Heroux for Sandia National Laboratories was parallelized with the qloop interface of the qthread library. The HPCCG program is a simple conjugate gradient benchmarking code for a 3-D chimney domain, largely based on code in the Trilinos[7] solver package. The code relies primarily upon tight loops where every iteration of the loop is essentially independent of every other iteration. This type of code is ripe for parallelization. With simple modifications to the code structure, the serial execution of HPCCG was transformed into multithreaded code. As illustrated in Figure 3.1, the parallelization is able to scale well. The performance numbers in this graph were obtained by running the code on a 48-node SGI Altix SMP.

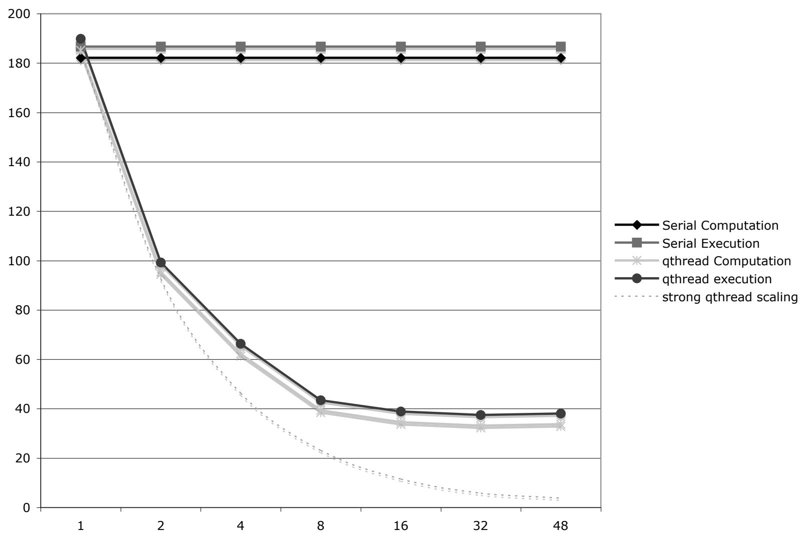


FIG. 3.1. *HPCCG Benchmark on a 48-node SGI Altix SMP*

**4. Conclusions.** Large scale computation of the sort performed by common computational libraries can benefit significantly from low-cost threading, as demonstrated here. Lightweight threading with hardware support is a developing area of research that the qthreads library assists in exploring while simultaneously providing a solid platform for lighter-weight threading on common operating systems. It provides basic lightweight thread control and synchronization primitives in a way that is portable to existing highly parallel architectures as well as to future and potential architectures. Because the API can be implemented and can provide MPI-like performance on existing platforms without a custom compiler, it allows study and modeling of the behavior of large scale parallel scientific applications for the purposes of developing and refining such parallel architectures.

## REFERENCES

[1] *Cray MTA-2 system - HPC technology initiatives*, November 2006.

[2] J. W. Berry, B. A. Hendrickson, S. Kahan, and P. Konecny, *Software and algorithms for graph queries on multithreaded architectures*, in Proceedings of the International Parallel & Distributed Processing Symposium, IEEE, 2007.

[3] J. B. Brockman, S. Thoziyoor, S. K. Kuntz, and P. M. Kogge, *A low cost, multithreaded processing-in-memory system*, in WMPI '04: Proceedings of the 3rd workshop on Memory performance issues, New York, NY, USA, 2004, ACM Press, pp. 16–22.

[4] L. Dagum and R. Menon, *OpenMP: An industry-standard API for shared-memory programming*, IEEE Computational Science & Engineering, 5 (1998), pp. 46–55.

[5] M. P. I. Forum, *MPI: A message-passing interface standard*, Tech. Report UT-CS-94-230, 1994.

[6] M. Hall, P. Kogge, J. Koller, P. Diniz, J. Chame, J. Draper, J. LaCoss, J. Granacki, J. Brockman, A. Srivastava, W. Athas, V. Freeh, J. Shin, and J. Park, *Mapping irregular applications to DIVA, a PIM-based data-intensive architecture*, in Supercomputing '99: Proceedings of the 1999 ACM/IEEE conference on Supercomputing (CDROM), New York, NY, USA, 1999, ACM Press, p. 57.

[7] M. Heroux, R. Bartlett, V. Hoekstra, J. Hu, T. Kolda, R. Lehoucq, K. Long, R. Pawlowski, E. Phipps, A. Salinger, et al., *An overview of Trilinos*, Tech. Report SAND2003-2927, Sandia National Laboratories, 2003.

[8] Institute of Electrical and Electronics Engineers, *IEEE Std 1003.1-1990: Portable Operating Systems Interface (POSIX.1)*, 1990.

[9] P. Kogge, S. Bass, J. Brockman, D. Chen, and E. Sha, *Pursuing a petaflop: Point designs for 100 TF computers using PIM technologies*, in Proceedings of the 1996 Frontiers of Massively Parallel Computation Symposium, 1996.

[10] C. P. Kruskal, L. Rudolph, and M. Snir, *Efficient synchronization of multiprocessors with shared memory*, in PODC '86: Proceedings fo the fifth annual ACM symposium on Prinicples of distributed computing, New York, NY, USA, 1986, ACM Press, pp. 218–228.

# RECONFIGURABLE FUNCTIONAL UNIT DESIGN FOR COMPLEX SCIENTIFIC DATAFLOW GRAPHS

KYLE RUPNOW[*] AND KEITH D. UNDERWOOD[†]

**Abstract.** Reconfigurable Functional Units (RFUs) tightly coupled with the main processor have significantly less communication overhead than a coprocessor model, allowing them to efficiently accelerate smaller computation graphs. However, tightly coupled RFUs are constrained by register file limitations, and the communication model of conventional functional units makes assumptions that are often invalid for RFUs. One graph execution using the RFU may require many inputs and each of the many outputs may complete at a different time. Furthermore, each execution of that same graph may generate a different number of needed outputs depending on the execution context. Therefore, we present a method that supports individual output bypassing and a variable and potentially large number of outputs for RFU instructions. To ensure that instructions do not wait to receive an operand that is already ready, we associate a ready bit with each of the resources in the RFU so that each output can complete and bypass to consuming instructions separately. We support a variable number of outputs by storing all potential graph outputs in a shadow register file. A compiler can track values stored in the shadow register file and insert move instructions to copy the values to the architected register file as needed. Thus, we generate all outputs and only consume register file bandwidth when an operand will be used. We have achieved an average 0.9% speedup with individual output bypassing, and 2.7% speedup with individual output bypassing plus a shadow register file.

**1. Introduction.** Many modern supercomputers are used as capability machines. The modern generation of capability supercomputers such as the Cray XT3 and BlueGene/L contains tens of thousands of processor cores, and it is not uncommon for the entire supercomputer to be devoted to one application for weeks at a time. In scientific computing the trend towards more processing cores faces significant challenges. It is already difficult for an application to efficiently utilize all of the available processing cores [10]. Load imbalance and communication overhead create particular challenges; parallel application developers expend significant effort to enable dynamic load balancing across cores in a large system while attempting to minimize the total required amount of communication (two competing goals). These overheads can limit the effective exploitation of parallelism by negating the benefit of additional computation threads. Thus, for many systems that already leverage hundreds of processing nodes, adding more processing cores per node cannot always provide the required performance improvement. In these cases, increasing individual core performance can provide an overall performance improvement without increasing communication overhead. Unfortunately, it has become increasingly difficult to improve single-threaded performance. Traditional techniques such as better branch prediction, larger instruction and data caches, larger issue queue and issue width, and more functional units often require corresponding increases in register file bandwidth and many of these structures are already on the processor's critical path. Recent research has suggested split cache structures [4], clustered architectures [17], dependence based scheduling [13], speculative wakeup [20] and macro-op scheduling [12] to enable effectively larger structures while minimizing the impact to the critical path.

Reconfigurable Functional Units (RFUs) can achieve many of the benefits of these approaches. A graph of potentially many processor instructions is extracted and compressed into a small number of RFU opcodes, giving the processor an effectively larger issue queue. Additionally, because issuing a single RFU opcode can represent the issue of multiple formerly separate instructions that may operate in parallel, issue width is effectively increased. The RFU configuration represents a pre-computed instruction schedule that describes both issue parallelism and any sequential dependence of graph instructions, similar to dependence based scheduling. The RFU also has a flexible routing network with localized communication

[*]University of Wisconsin-Madison,kjrupnow@wisc.edu
[†]Intel Corp., kdunder@sandia.gov

paths similar to those of clustered architectures, so connections between functional resources within the RFU can be optimized for the instruction graph.

Most prior work involved augmenting relatively simple processor architectures with reconfigurable hardware. The complexity of processors used in scientific computing presents additional challenges to RFU design for these devices. An instruction graph may require many inputs and produce many outputs; however, not all inputs may be needed simultaneously for the graph to begin useful computation, and not all outputs may be generated at the same time. Although the RFU could simply wait for all inputs to be available before beginning execution, this could result in worse application performance than a baseline superscalar processor that uses out-of-order execution to mitigate variable input arrival latencies. This paper presents an RFU architecture augmented with "ready bits" to allow the RFU to compute as inputs are available. Another issue faced by RFUs is that an application may contain the same instruction graph in multiple areas of the code, but require different sets of outputs from the different instances of the graph. A simple solution is to ensure that the RFU commits all unique register destinations in the graph to architected state. This can result in unnecessary communication in many cases, slowing overall performance. Therefore, we also present a new RFU architecture that uses both ready bits and a shadow register file to efficiently handle variability in quantity and timing of inputs and outputs.

**2. Related Work.** Many architectures have used reconfigurable hardware to accelerate applications [9, 22, 23, 1, 8, 18, 5, 6, 19]. Fine-grained reconfigurable logic efficiently accelerates bit-level manipulations [8, 18, 23], whereas coarse-grained architectures use common compute units such as ALUs and multipliers to accelerate word-sized computations [6, 19, 5]. Fine-grained architectures provide greater flexibility than coarse-grained ones, but at the expense of increased configuration data and thus a longer reconfiguration time.

Communication methods between reconfigurable hardware and a host processor is affected by the coupling between the two. Two of the main coupling methods include a functional unit model and a coprocessor model. Reconfigurable functional units (RFUs), such as [8, 18, 5] and those examined by Rupnow et al [19] communicate with the processor through the register file. This can minimize communication overhead for operations with a small number of operands, and allow the RFU to efficiently accelerate smaller instruction graphs than possible with a coprocessor model. A reconfigurable coprocessor may receive some data from the host processor, but also communicates directly with other resources, such as memory or I/O [9, 22, 23, 1]. Operations mapped to a reconfigurable coprocessor frequently are very regular computations performed repeatedly over a great deal of regularly-spaced (strided) data. The coprocessor model allows reconfigurable hardware to perform more complex computations for longer periods of time than a functional unit model, but incurs a greater startup time.

This work targets integer operations within scientific computing applications. These operations are word-sized, and tend to require complex memory access patterns unsuited for stream-style computation. Therefore, we focus on an RFU model with a coarse-grained internal structure. The coarse-grained structure will also permit the RFU to be reconfigured more quickly between different instruction graphs. However, this paper focuses primarily on the high-level interaction of the RFU with the processor, leaving the exact structure of the coarse-grained reconfigurable fabric to future efforts.

Most groups limit the number of input and output operands to their RFUs, frequently either to ensure that an operation can be expressed within a single opcode, or to prevent the port count of the register file from expanding beyond a reasonable quantity. This constraint effectively limits the size (latency) of the instruction graphs implementable by the RFU, but also avoids the problem of differing output completion times. Another group [19] found that a significant number of their targeted graphs have a large number of outputs, thus they do not limit the input or output count. Instead, like the new RFU architectures presented in this paper, they use multiple RFU opcodes to encode a given graph to limit register file bandwidth. However, they write graph outputs to the register file based on a worst case latency estimate rather than when the output data is actually ready. Furthermore, because different outputs may be required from a graph based on when in the application it is used, they write all unique register destinations in the graph to architected state, whether or not those registers are subsequently used for the given graph instance.

Section 3.3 presents analysis showing that on average only 50% of intermediate graph values are used by later instructions, and for particular applications, as few as 25% are used. This significant number of unnecessary register writes represents a significant amount of wasted register file bandwidth. This paper proposes a new RFU architecture that uses a shadow register file (SRF) to locally store graph outputs. A compiler tracks live values assigned to the SRF, and moves them to the architected state if later instructions will use that data. A compiler could instead create a different RFU configuration for each case of different output requirements. However, this would reduce configuration reuse, which in turn increases configuration storage requirements and reconfiguration overhead penalties.

Previous register allocation schemes use techniques similar to this SRF approach to determine which register values need to be committed to architected state. Moudgill et. al [16] proposed a renaming algorithm that tracks whether a register name is the active owner of an architected register, and how many consumers of that value have not yet received the value. More recently, groups have proposed using this information to release the physical register early once they have determined the register value will never again be used [2]. Other techniques include specifically detecting operands that are only used once [11], tracking the reorder buffer for subsequent writes to the same architected register without an intervening read [14], and late allocation schemes that only allocate a physical register once the functional unit generates the data and it is determined that there will be a consuming instruction [15]. These methods must decide whether to commit a value to architected state by the time the value reaches the head of the reorder buffer.

The SRF used with this RFU will instead store data separately from the reorder buffer, holding it until a subsequent write invalidates it or the data is copied to architected state. The SRF also serves as local storage a different RFU graph uses a value currently stored in the SRF, it will use the value directly out of local storage rather than forcing the value to propagate through architected state.

**3. Method.** The new RFU architectures were evaluated using trace-based simulation of an out of order execution model and a set of graphs selected from the target applications using trace-based analysis. The simulations use the Structural Simulation Toolkit (SST) [21], which couples a PowerPC trace [7] input with a SimpleScalar-based out of order execution model [3]. SST supports multiple processors, a realistic memory hierarchy. Although SST uses traces, it emulates of out of order execution with cache misses and wrong path execution. The simulator scans each trace to generate a memory map of the application, which it then uses to provide instructions to the out of order execution model. For normal execution, the trace translation tracks the next correct instruction to execute. For wrong-path execution, it provides the requested instruction from the memory map, with the instruction marked as "fake" so that the trace interpreter and execution model know that the instruction is part of wrong-path execution. This method provides a balance between determinant trace simulation and emulation of wrong-path execution impacts seen in execution-based simulation.

The RFU architectures accelerate graphs previously selected using trace based analysis. The trace analysis uses a greedy algorithm to generate and select graphs for RFU execution based on the size of the graphs, the number of times they are executed, and the percentage of trace instructions they cover. Each application trace contains four billion instructions selected by source code analysis and performance profiling to cover the "primary loop" of the application. We use 32 total traces, of which 17 are traces of real scientific applications in production use at Sandia National Laboratories and 15 are traces of the SPEC-FP benchmark suite. The individual benchmarks are described in Table 3.

Table 3.1
*Benchmark Description*

| **Suite** | **Benchmark** | **Description** |
|---|---|---|
| Sandia | Alegra | Shock physics |
| Sandia | Cth | Shock physics |
| Sandia | cube3 | Sparse Matrix Solver |
| Sandia | Its | Radiation transport |
| Sandia | lammps | Molecular dynamics |
| Sandia | Mpsalsa | Chemically reactive flow |
| Sandia | Xyce | Circuit Simulation |
| SPEC-FP | 168.wupwise | Physics/Quantum Chromodynamics |
| SPEC-FP | 171.swim | Shallow Water Modeling |
| SPEC-FP | 172.mgrid | Multi-grid Solver: 3D Potential Field |
| SPEC-FP | 173.applu | Parabolic/Elliptic PDE |
| SPEC-FP | 177.mesa | 3D Graphics Library |
| SPEC-FP | 178.galgel | Computational Fluid Dynamics |
| SPEC-FP | 179.art | Image Recognition / Neural Network |
| SPEC-FP | 183.equake | Seismic Wave Propagation |
| SPEC-FP | 187.facerec | Image Processing: Face Recognition |
| SPEC-FP | 188.ammp | Computational Chemistry |
| SPEC-FP | 189.lucas | Number Theory / Primality Testing |
| SPEC-FP | 191.fma3d | Finite-element Crash Simulation |
| SPEC-FP | 200.sixtrack | High Energy Nuclear Physics Accel. |
| SPEC-FP | 301.apsi | Meteorology: Pollutant Distribution |

Table 3.2
*Baseline Processor Configuration*

| | |
|---|---|
| RUU Size | 64 |
| LSQ Size | 32 |
| Fetch Width | 4 |
| Issue Width | 8 |
| Commit Width | 4 |
| ALU's | 3 |
| Multipliers | 1 |
| Memory Ports | 2 |
| FP ALU's | 2 |
| FP Multipliers | 1 |
| Instruction Cache | 512K |
| Data L1 | 32K |
| L2 | 2M |

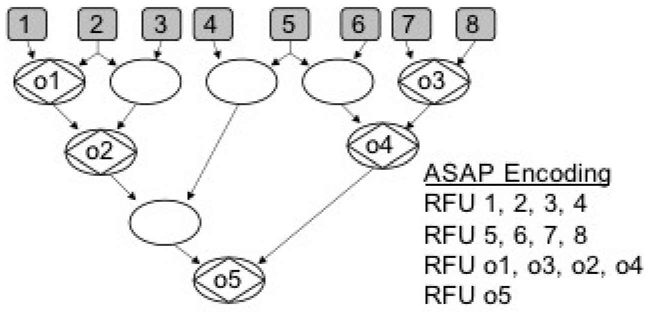


FIG. 3.1. *ASAP Scheduling*

**3.1. RFU Architectures.** We compare two sets of RFU architectures: the first set has ready bits to allow intermediate output bypassing, and the second has ready bits plus a shadow register file (SRF). The baseline architecture information is given in Table 3. Both of the RFUs are coarse grained fabrics of add/subtract/logic and multiply/shift/rotate units connected via a feed forward routing network, with input and output buffers. For each of these RFUs, we test four variations with different maximum limits on the number of operands per opcode. The operand limit constrains the total number of inputs and outputs per opcode, but does not constraint how many operands within the limit are inputs vs. outputs. Maximums of four, five, or six operands are intended to represent a reasonable limit on register file bandwidth and instruction encoding. A maximum of fifteen total operands provides an extreme upper bound on allowable register file bandwidth and instruction encoding.

Graph inputs and outputs are encoded in opcodes in the order encountered, based on the premise that "early" inputs or outputs should be encoded before "late" ones. However, graph scheduling affects this order. For example, an as late as possible (ALAP) scheduling may cause an input use to occur much later in the ordering than an as soon as possible (ASAP) scheduling. ASAP may be preferable if all inputs are available when the RFU instruction begins execution, to maximize parallelism of graph computation. On the other hand, ALAP may perform better if, for example, an RFU input is not yet ready, but is also not needed until later in the graph execution.

Tables 3.1 and 3.2 show a comparison of the effects of ASAP and ALAP scheduling on instruction encoding with a four-operand-per-opcode limit. In Table 3.1, the first two opcodes encode all 8 inputs, and the remaining two opcodes encode the five outputs in the order they would complete. Communication between the RFU opcodes of a single instruction graph is encoded as part of the RFU configuration, and is not encoded in those opcodes. ASAP scheduling moves all nodes with only external inputs to the top of the graph. Because of this ordering, all input operands in this example will be encoded first, followed by outputs in the order they complete. Thus, ASAP scheduling will likely have opcodes with only outputs, which can issue immediately because they only provide output register numbers. Conversely, ALAP scheduling will tend to interleave input and output operands so that inputs and related outputs will often be encoded together.

**3.2. Ready Bits.** Ready bits are essentially flags associated with individual or groups of compute elements in a larger structure. A true value indicates that the associated compute structure has received its inputs and completed its computation. A false value indicates that either the inputs are not yet available, or they are available but the computation is not yet complete. Therefore, ready bits allow data to flow through multiple connected compute structures as their inputs become available, and indicate when the overall computation out-

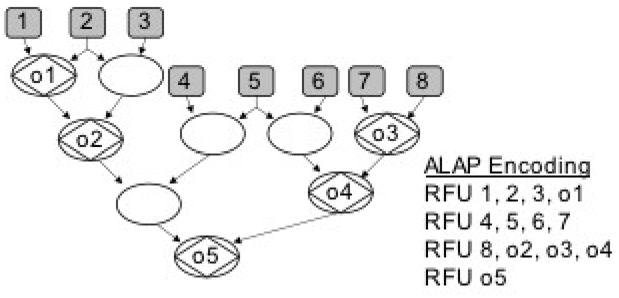


FIG. 3.2. *ALAP Scheduling*

puts are complete. Dataflow processors use a similar approach, but generally communicate through FIFO queues instead of a local routing network. When all inputs to a processing element are ready, that element performs its computation and asserts the ready bit to place the result into the queue(s) of the consumer(s). The ready bit prevents invalid data from entering the queue due to a non-constant or even non-deterministic number of clock cycles between outputs. However, the ready bit also allows computation to proceed as quickly as possible, as the consumer does not have to always wait a worst-case number of cycles before it can safely assume valid input data is present.

Each entry in the RFU's the input buffer contains a ready bit, and each of the RFU's resources also contains an AND gate which combines the ready bits of the resource's input operands. RFU opcodes may be issued out of order, providing inputs at different times; the ready bits indicate when the inputs have filtered through and graph execution is complete. The RFU is considered busy (and unable to reconfigure or execute a new graph) until the ready bits propagate to all of the outputs. Figure 3.3 illustrates an example using ready bits. Numbered squares indicate register inputs to the graph, and diamonds in nodes indicate graph outputs. Nodes not marked as outputs represent instructions that output to registers overwritten later in the graph. Input and output nodes are numbered to identify unique operands. The first RFU opcode is waiting for its input operands to be ready, and the following three opcodes have already issued out-of-order. The ready bits for inputs 4-8 are set and results propagate through the shaded instructions. In this case, the instructions that generate o3 and o4 are complete, and both of those outputs are allowed to bypass to consuming instructions while the rest of the graph waits for the final three inputs.

**3.3. Shadow Register File Architecture.** The shadow register file (SRF) is internal storage for the RFU. RFU opcodes inform the RFU which architected registers are potential graph outputs based on graph instructions writing values to unique registers not overwritten by later graph instructions. When the graph executes, the RFU stores graph results into the SRF instead of back to the architected register file. This means that RFU opcodes can effectively commit for free, and do not count against instruction limit or register file port limit. A compiler can track which potential graph outputs are resident in the SRF. If other RFU instructions use values currently stored in the SRF, then they may use the value directly instead of retrieving it from the architected register. Otherwise, if a non-RFU instruction needs a value stored in the SRF, the compiler inserts move instruction to transfer the value from the SRF to the architected register file. Because the RFU instructions no longer write data to architected state, they can commit and exit the reorder buffer. However, the penalty is the need for inserted move instructions, which are blocked by the RFU ready bits until the needed value is ready. Moves from the SRF are destructive, so subsequent reads will take

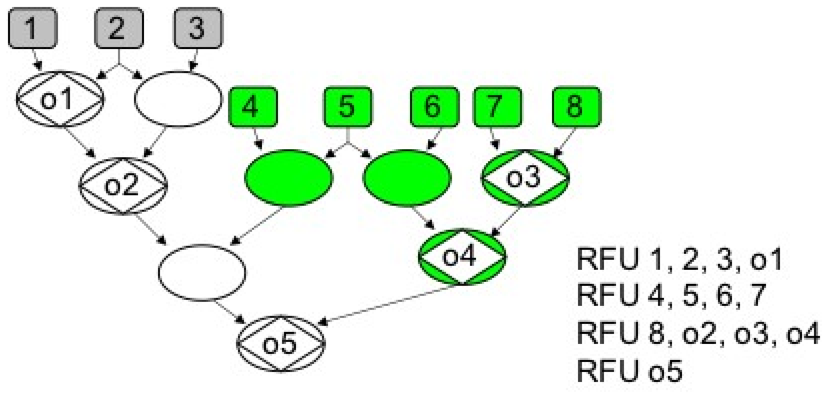


FIG. 3.3. *Ready Bits in RFU Graph Execution. The first opcode is waiting for inputs to be ready. The following three opcodes have already issued out of order.*

the value from the architected register (avoiding coherence problems). SRF locations are also invalidated if another non-RFU instruction writes to the same architected register, as they will no longer contain the most current data for that register. By only sending data values that are needed to the architected register file, we can reuse RFU configurations and efficiently handle a large and potentially variable number of outputs while minimizing the register file bandwidth used by the RFU.

The trace analysis portion of the SST simulator emulates SRF operation. First, the simulator finds sets of instructions belonging to graphs chosen for acceleration. Next, those instructions are replaced by a sequence of RFU opcodes that encode graphs operands using the architecture's operand limit. Trace analysis keeps track of the architected register numbers held in the SRF, and inserts special move opcodes before subsequent reads of those registers. In the execution model portion of SST, the commit operation detects RFU opcodes and commits them for free.

Because the SRF only tracks a single value per architected register, write after write data hazards could be a potential problem if RFU opcodes are issued out of order. However, this is only the case for RFU opcodes from differing instruction graphs, since only final writes to registers are considered outputs, and will write to the shadow register file. Intermediate writes to those registers instead are identified as internal RFU communication. To prevent write after wrote data hazards, the RFU must only contain a single instruction graph, and applications cannot resuse this graph until it is done with the current iteration of the instruction graph. When the first RFU opcode for a given graph issues, it allocates the RFU, and the RFU will remain busy until the entire graph of computation completes. This prevents out of order completion of RFU graphs, which also prevents write after write data hazards.

**4. Results.** We present performance results from an RFU architecture that includes ready bits followed by register usage analysis to motivate the use of the shadow register file, and then compare to an RFU architecture with ready bits and a shadow register file. Application speedups are compared for both architectures, along with instruction fetch queue (IFQ), register update unit (RUU) and load/store queue (LSQ) occupancy. Finally, we compare application speedup and average number of opcodes needed to encode a graph to previously published results [19].

**4.1. RFU Architecture: Ready Bits.** The RFU architecture with ready bits achieves up to 8.8% speedup on the applications studied. Overall, the RFU accelerated 26 of the 32 appli-

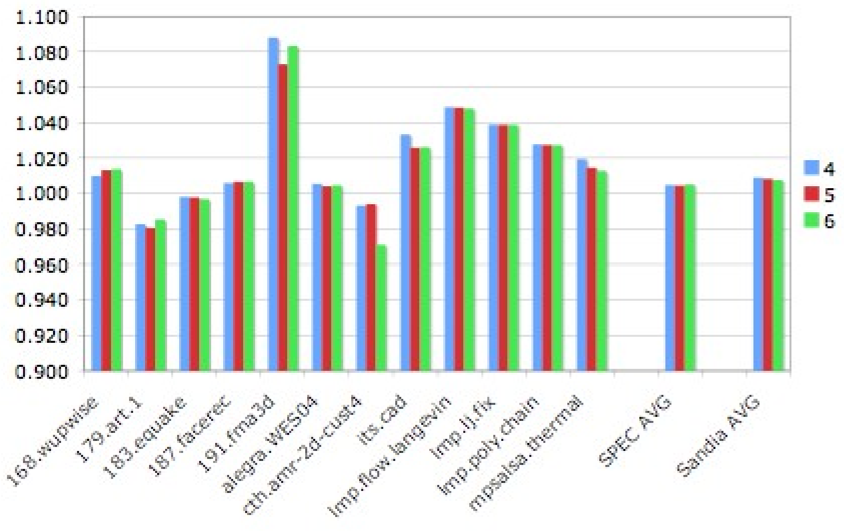


FIG. 4.1. *Ready Bits Architecture Speedup Over Baseline by Number of Operands per Opcode (Selected Applications)*

cations, some by as much as 8.8%. Although 6 of 32 traces had slowdown, four were slowed less than 1%, and none more than 1.7%. Although only 2 of the 15 SPEC-FP traces achieve 2% or more speedup, 8 of the 17 real scientific application traces achieve 2% or more speedup for at least one of the three tested operand limits. Figure 4.1 shows the speedup of selected applications over the baseline architecture for varying numbers of maximum operands per opcode from both benchmark suites, as well as the average performance for all SPEC-FP and all Sandia applications. In many of the cases, increased operands per opcode degrades performance because the opcodes cannot commit until all of the outputs in the opcode complete. The idealized 15 operand per opcode limit ready bits architecture shows the same effect. In many cases it is slower than the other architectures because the entire graph is encoded in one opcode, and thus cannot commit until the entire graph execution is complete.

**4.2. RFU Architecture: Ready Bits + Shadow Register File.** Although graphs may have a large number of potential outputs (uniquely written intermediate registers), a significant percentage of these may not be used by subsequent instructions. However, because different uses of a particular graph of instructions in different areas of the application may use different outputs, they cannot necessarily be eliminated as potential outputs. Five of the seventeen Sandia traces use fewer than 30% of potential graph outputs, and on average Sandia applications use only 52% of the potential graph outputs. Some applications do not tend to use many of the possible graph outputs, others use almost all graph outputs. Two SPEC-FP benchmarks, 173.applu and 178.galgel, both use over 95% of potential graph outputs. Figure 4.2 shows the average number of graph outputs used per application and averages for the SPEC-FP benchmark suite and Sandia applications.

Adding an SRF to the architecture with ready bits yields up to a 13.9% speedup over baseline, and 11.8% over ready bits alone, with all applications accelerated to some degree. These speedup values account for the overhead of the added move instructions. Some of the best application speedups are on the applications that use a low percentage of graph outputs. Application speedups are within 1% of the idealized 15 operand limit SRF architecture in

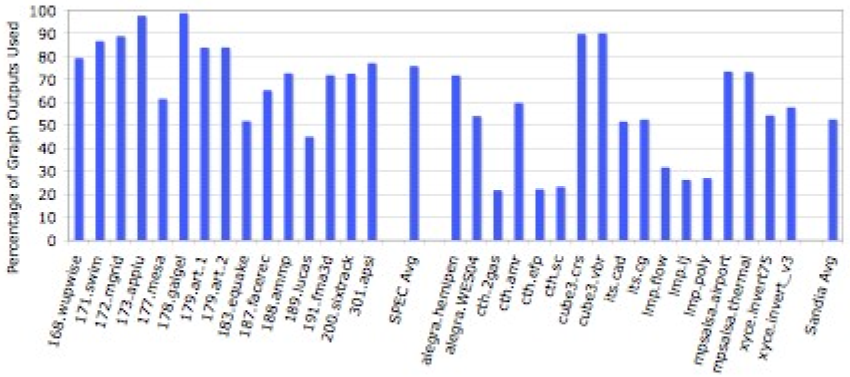


FIG. 4.2. *Percentage of Potential Graph Outputs Actually Used*

all but two cases. These two traces, lmp.flow and lmp.lj, are 2.2% and 1.3% worse than the 15 operand limit SRF respectively. Increasing the operand count to the idealized 15 operand limit in the SRF case does not degrade performance because, with the SRF, the opcodes still commit for free. Thus, the 15 operand SRF architecture is always equal to or better than the smaller operand limits because it can take advantage of increased encoding efficiency without causing the opcodes to wait longer to commit. As might be expected, the 15 operand limit SRF architecture performance is identical for ASAP encoding and ALAP encoding because the graphs are encoded in a single instruction.

Although in the architecture with ready bits but no SRF, ALAP scheduling achieves trivially better speedup than ASAP (less than 0.5% difference), with the SRF, ASAP scheduling achieves as much as 4% better speedup than ALAP. In the ready bits architecture, opcodes with interleaved operands can improve performance because an operand with a mixture of inputs and outputs may have less outputs to wait to commit. The architecture with a SRF removes that sensitivity because the RFU opcodes no longer wait for outputs to commit. Instead, when the graph performance is sensitive to the encoding at all, it benefits from having inputs encoded first to enable the architecture to begin computation as quickly as possible. Furthermore, opcodes with only outputs can issue immediately because no inputs are present in the opcode that can block issue. ALAP encoding order of operands does not necessarily correspond to the timing of when input values will actually be ready, so interleaved inputs and outputs can slow issue of output information waiting for late arriving inputs.

Figures 4.3 and 4.4 compare the maximum speedup achieved by any ready bits architecture without SRF to the maximum speedup for the ready bits plus SRF architecture for SPEC-FP and Sandia applications respectively. Although SPEC-FP applications show relatively small performance improvements by adding an SRF to the ready bits support, all of the applications that slowed by the ready bits architectures achieve speedup with the SRF. In contrast, nearly every Sandia application achieves significantly higher performance with the SRF, which correlates to the fact that Sandia averages fewer outputs used per graph, and so Sandia applications save more by writing only the needed outputs to the architected register file. For the architecture with ready bits but no SRF, encoding more operands per opcode provides little incremental benefit. In fact, Figure 4.1 showed that in many cases the application performance decreases when additional operands are encoded per opcode. Although graph outputs can bypass to consumers as soon as they complete, the opcode cannot commit until all outputs for that opcode complete. Opcodes with many outputs may therefore wait longer

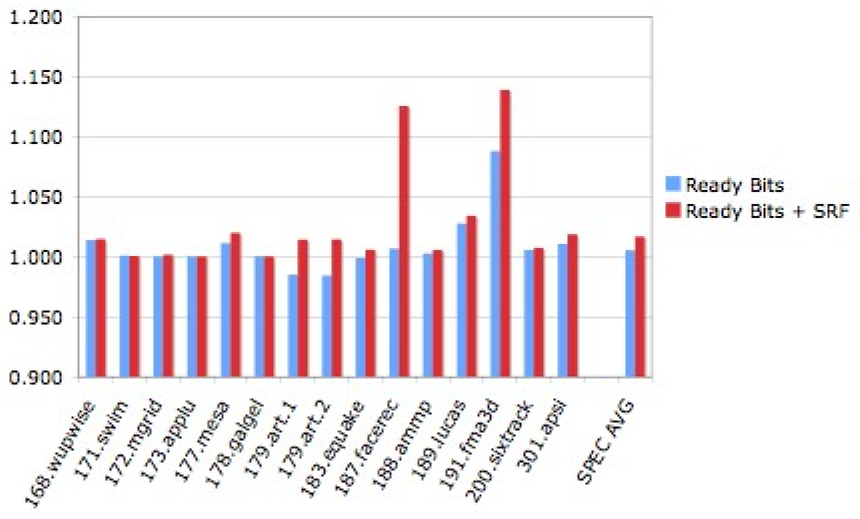


FIG. 4.3. *Maximum Application Speedup - Ready Bits vs. Ready Bits + SRF (SPEC-FP)*

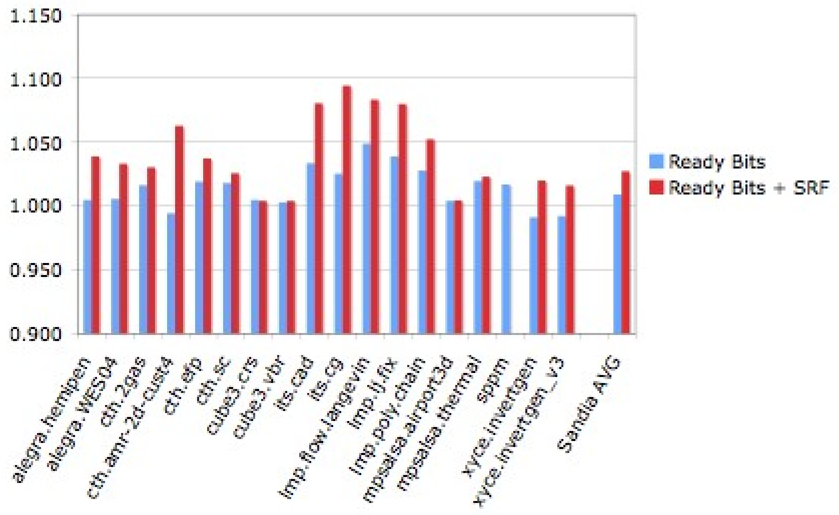


FIG. 4.4. *Maximum Applications Speedup - Ready Bits vs. Ready Bits + SRF (Sandia)*

to commit than opcodes with fewer outputs, degrading performance. In contrast, because the SRF architectures do not wait on output readiness to commit RFU opcodes (regardless of the number of operands) their performance improves as the number of operands per opcode increases.

Both architectures cause a decrease in average RUU occupancy that ranges from 1% in the common case to as much as 10% in specific cases. The SRF architecture achieves lower average RUU occupancy than the ready bits architecture without SRF despite a larger total number of instructions (the inserted move instructions). Because the RFU instructions commit for free (they do not write values to the architected register file), they spend less time in the RUU, thus freeing space for other register updates. Similarly, both RFU architectures cause an average decrease in IFQ occupancy. However, although the architecture with a shadow register file on average decreases LSQ occupancy, the architecture with only ready bits on average increases LSQ occupancy. Because store instructions acquire a memory port at commit, the amount of time they spend in the LSQ is determined based on how quickly they commit. In the ready bits architecture, long latency RFU operations may delay commit at the head of the RUU, thus delaying later store operations. In contrast, RFU operations in the SRF architecture will commit quickly, reducing the delay on later store operations and thus LSQ occupancy.

**4.3. Comparison to Previous Data.** Previously published results demonstrate speedups between 0.2% and 22.6% across a range of applications [19]. However, that architecture also negatively impacted 12 of the 32 traces with slow downs of as much as 6.3%. The previous work also assumed a simplistic processor model. The execution model used in this paper is more sophisticated, and includes wrong path execution, a realistic instruction cache, out of order issue for the RFU instructions, individual output bypassing for each of the RFU's outputs, and operand flexible instruction encoding. The ready bits architecture accelerates applications by 0.2% to 8.8% and slows only six applications by at most 1.8%. The SRF architecture achieves speedups between 0.3% and 13.9%, with an average speedup of 1.5% for SPEC-FP benchmarks and 2.7% for Sandia applications. Furthermore, no applications were slowed by the RFU with a SRF. Figure 4.5 shows six of the best speedups over the previously published data, the three worst slowdowns, and the average over the entire SPEC-FP benchmark suite and the Sandia applications. For each of these measurements, the figure displays the best application performance over all tested architectures. For the three SPEC-FP benchmarks shown, the SRF architecture outperforms the best case previously published result by 6.5-15.3%.

Although the three best performing applications from the prior results perform significantly better than the new architectures, the SRF architecture still achieves speedup between 5.2% and 8.3% for those three applications. One possible reason for better results in the previous work is that they used an idealized architecture (for both the baseline and RFU architectures) that assumes a perfect instruction cache and prevents wrong path execution. The model used for this study instead allows wrong path execution and models a realistic instruction cache.

The previously published operand encoding algorithm is not directly comparable to our current method's although their maximum number of operands is similar, they limit inputs and outputs separately, and never allow more than four outputs in an opcode. In contrast, the proposed encoding method limits only the total operand count, providing more flexibility to the encoder. The proposed encoding method will not only always fill opcodes with operands when possible, but will also likely encode more outputs than allowed previously given that the previous output limits were often lower than the new total operand limits. Approximately 25% fewer opcodes are required to encode an RFU operation in the new architectures versus the previous, and the new architectures often outperform the previous even when the previous can encode a larger number of operands per opcode. This is due to two causes. First, the new architectures can better fill each opcode because inputs and outputs are not separately limited. Second, in the new architecture, the output encoding can be provided before the output is ready, whereas the prior model provided output encoding only after the output will be ready. This additional flexibility allows us to guarantee that each graph will be encoded in the minimum number of opcodes given the operand limit per opcode.

**5. Conclusion.** In real scientific applications, kernels of computation may have large numbers of both input and output operands. It is important that new RFU architectures can

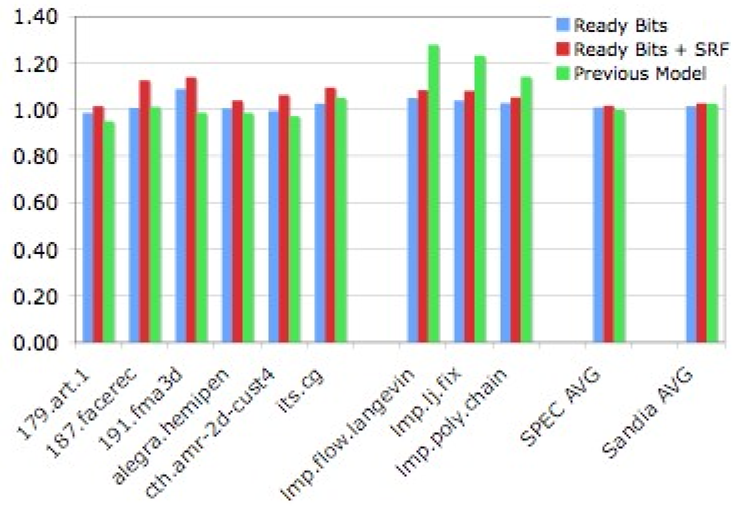


FIG. 4.5. *Average Application Speedup*

accelerate the complex graphs of computation present in modern applications. Therefore, RFU architectures need to efficiently handle many inputs and a large and potentially variable number of outputs. This paper presented an RFU architecture that meets these requirements by using multiple opcodes to collaboratively describe a graph, using ready bits to allow outputs to complete and bypass to consuming instructions individually, and most importantly, using a shadow register file (SRF) to support a large number of outputs yet minimize the number of writes to the architected register file. The proposed RFU architecture with an SRF accelerates all of the applications, with speedups of up to 13.9% over the baseline architecture and 11.8% over an RFU architecture with ready bits but no SRF. Furthermore, SRF-based architectures with four, five or six maximum operands per opcode achieve at least 98% of the speedup of the idealized 15 operand limit, and in most cases achieve speedups within 0.1% of the 15 operand limit. Thus, the new RFU architecture with ready bits and an SRF is able to effectively accelerate production applications that exhibit complex program behavior.

## REFERENCES

[1] J. M. Arnold, *S5: The architecture and development flow of a software configurable processor.*, in IEEE International Conference on Field Programmable Technology, Dec 2005, pp. 121–128. crossref: DBLP:conf/fpt/2005.

[2] D. Balkan, J. Sharkey, D. Ponomarev, and K. Ghose, *Selective writeback: exploiting transient values for energy-efficiency and performance*, in ISLPED '06: Proceedings of the 2006 international symposium on Low power electronics and design, New York, NY, USA, 2006, ACM Press, pp. 37–42.

[3] D. C. Burger and T. M. Austin, *The simplescalar tool set, version 2.0*, Tech. Report CS-TR-97-1342, University of Wisconsin, Madison, 1997 1997.

[4] B. Calder, D. Grunwald, and J. Emer, *Predictive sequential associative cache*, 1996, pp. 244–253.

[5] N. Clark, J. Blome, M. Chu, S. Mahlke, S. Biles, and K. Flautner, *An architecture framework for transparent instruction set customization in embedded processors*, in ISCA '05: Proceedings of the 32nd International Symposium on Computer Architecture, 2005, pp. 0–12.

[6] D. C. Cronquist, C. Fisher, M. Figueroa, P. Franklin, and C. Ebeling, *Architecture design of reconfigurable pipelined datapaths*, in Advanced Research in VLSI, 1999. Proceedings. 20th Anniversary Conference on, 1999, pp. 23–40.

[7] A. A. P. Groups, *Computer Hardware Understanding Development Tools 2.0 Reference Guide for MacOS X*,

Apple Computer Inc, July 2002.

[8] S. Hauck, T. W. Fry, M. M. Hosler, and J. P. Kao, *The Chimaera reconfigurable functional unit*, in FCCM '97: Proceedings of the 5th IEEE Symposium on FPGAs for Custom Computing Machines, IEEE Computer Society, 1997, p. 87.

[9] J. R. Hauser and J. Wawrzynek, *Garp: A MIPS processor with a reconfigurable coprocessor*, in FCCM '97: Proceedings of the 5th IEEE Symposium on FPGAs for Custom Computing Machines, IEEE Computer Society Press, 1997, pp. 12–21.

[10] A. Hoisie, G. Johnson, D. J. Kerbyson, M. Lang, and S. Pakin, *A performance comparison through benchmarking and modeling of three leading supercomputers: Blue Gene/L, Red Storm, and Purple*, in SC '06: Proceedings of the 2006 ACM/IEEE conference on Supercomputing, New York, NY, USA, 2006, ACM Press, p. 74.

[11] T. M. Jones, M. F. R. O'Boyle, J. Abella, A. Gonzalez, and O. Ergin, *Compiler directed early register release*, in Parallel Architectures and Compilation Techniques, 2005. PACT 2005. 14th International Conference on, 2005, pp. 110–119.

[12] I. Kim and M. H. Lipasti, *Macro-op scheduling: Relaxing scheduling loop constraints*, in MICRO 36: Proceedings of the 36th annual IEEE/ACM International Symposium on Microarchitecture, IEEE Computer Society, 2003, p. 277.

[13] P. Michaud and A. Seznec, *Data-flow prescheduling for large instruction windows in out-of-order processors*, in Proceedings of the Seventh International Symposium on High-Performance Computer Architecture (HPCA'01), IEEE Computer Society, 2001, p. 27.

[14] T. Monreal, V. Vinals, A. Gonzalez, and M. Valero, *Hardware schemes for early register release*, in Parallel Processing, 2002. Proceedings. International Conference on, 2002, pp. 5–13.

[15] T. Monreal, V. Vinals, J. Gonzalez, A. Gonzalez, and M. Valero, *Late allocation and early release of physical registers*, Transactions on Computers, 53 (2004), pp. 1244–1259.

[16] M. Moudgill, K. Pingali, and S. Vassiliadis, *Register renaming and dynamic speculation: an alternative approach*, in Microarchitecture, 1993. Proceedings of the 26th Annual International Symposium on, 1993, pp. 202–213.

[17] S. Palacharla, N. P. Jouppi, and J. E. Smith, *Complexity-effective superscalar processors*, 1997, pp. 206–218.

[18] R. Razdan and M. D. Smith, *A high-performance microarchitecture with hardware-programmable functional units*, in MICRO 27: Proceedings of the 27th Annual International Symposium on Microarchitecture, ACM Press New York, NY, USA, 1994, pp. 172–180.

[19] K. Rupnow, K. D. Underwood, and K. Compton, *Scientific application acceleration with reconfigurable functional units*, Procedings of the IEEE International Conference on Field Programmable Custom Computing Machines, (2007), pp. 1–10.

[20] J. Stark, M. D. Brown, and Y. N. Patt, *On pipelining dynamic instruction scheduling logic*, 2000, pp. 57–66.

[21] K. D. Underwood, M. Levenhagen, and A. Rodrigues, *Simulating Red Storm: Challenges and successes in building a system simulation*, in Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International, 2007, pp. 1–10.

[22] M. J. Wirthlin and B. L. Hutchings, *Sequencing run-time reconfigured hardware with software*, in FPGA '96: Proceedings of the Fourth ACM International Symposium on Field-Programmable Gate Arrays, ACM Press New York, NY, USA, 1996 1996, pp. 122–128.

[23] R. D. Wittig and P. Chow, *OneChip: an FPGA processor with reconfigurable logic*, in FCCM '96: Proceedings of the IEEE Symposium on FPGAs for Custom Computing Machines, 1996, pp. 126–135.

# ARCHITECTURAL EXTENSIONS FOR EXECUTING FLOATING POINT INSTRUCTION AGGREGATES

PATRICK LA FRATTA*, ARUN RODRIGUES†, AND KEITH D. UNDERWOOD‡

**Abstract.** Floating point computation comprises a large portion of execution time in many modern applications, such as scientific simulation codes run at Sandia National Laboratories. In a conventional superscalar, out-of-order architecture, each floating point operation is executed as a single instruction. This work explores an alternative execution model in which groups of dependent floating point operations are encoded into a single instruction, called a *floating point instruction aggregate* (FPIA). We present the requirements of an architecture to support such instructions, and estimate the potential performance improvements offered by such an architecture through simulation. The results show improvements of over 7% in execution time.

**1. Introduction.** Many modern applications demand increasing levels of performance from supercomputers, and a large number of these spend a significant amount of execution time performing floating point (FP) computation. For example, at Sandia National Laboratories, simulation codes that seek to solve several scientific problems have been shown to contain instruction mixes with up to 25% FP computation [8].

In a conventional superscalar processor supporting out-of-order (OoO) execution, each instruction executed in the pipeline requires a certain amount of overhead in execution time. For example, each instruction consumes resources (such as in the issue queue and reservation stations) and must undergo dependence checking.

This work explores an approach to reducing this overhead for floating point instructions by encoding dependent floating point operations into a single instruction prior to run-time. The resulting multi-operation instruction is called a *floating point instruction aggregate* (FPIA). This paper presents an architecture for executing FPIAs, and analyzes the content of applications to determine the properties of a typical FPIA. The results from these analyses are used to set design parameters for the architecture. We then estimate the potential performance improvement of this architecture over conventional OoO processors. The performance estimate is obtained using trace-driven simulation, where the trace has been transformed to contain FPIAs.

The section that follows presents an overview of the FPIA architectural extensions. Section 3 gives our methodology used for FPIA design and evaluation. Section 4 offers a brief description of past work with instruction aggregates. Section 5 presents the details of graph analyses and corresponding results. Our approach to performance evaluation through simulation is given in section 6, along with the performance results. In section 7, we summarize and describe future tasks of interest.

**2. FPIA: A Conceptual Overview.** The goal of FPIA is to group FP operations prior to run-time, encoding all information necessary to execute these operations, including dependence information and register I/O, in a single instruction. The reasons for this are to both reduce single-instruction overhead incurred by each FP operation in the conventional OoO architecture and to expose parallelism prior to run-time. This simplifies the requirements of the OoO execution hardware.

The execution of these multi-operation instructions requires a special organization of floating point functional units (FUs). The organization considered here is a 2-dimensional array of FUs with the interconnect necessary to route operands among them and distribute

*University of Notre Dame, plafratt@nd.edu
†Sandia National Laboratories, afrodri@sandia.gov
‡Intel Corporation, kdunder@gmail.com

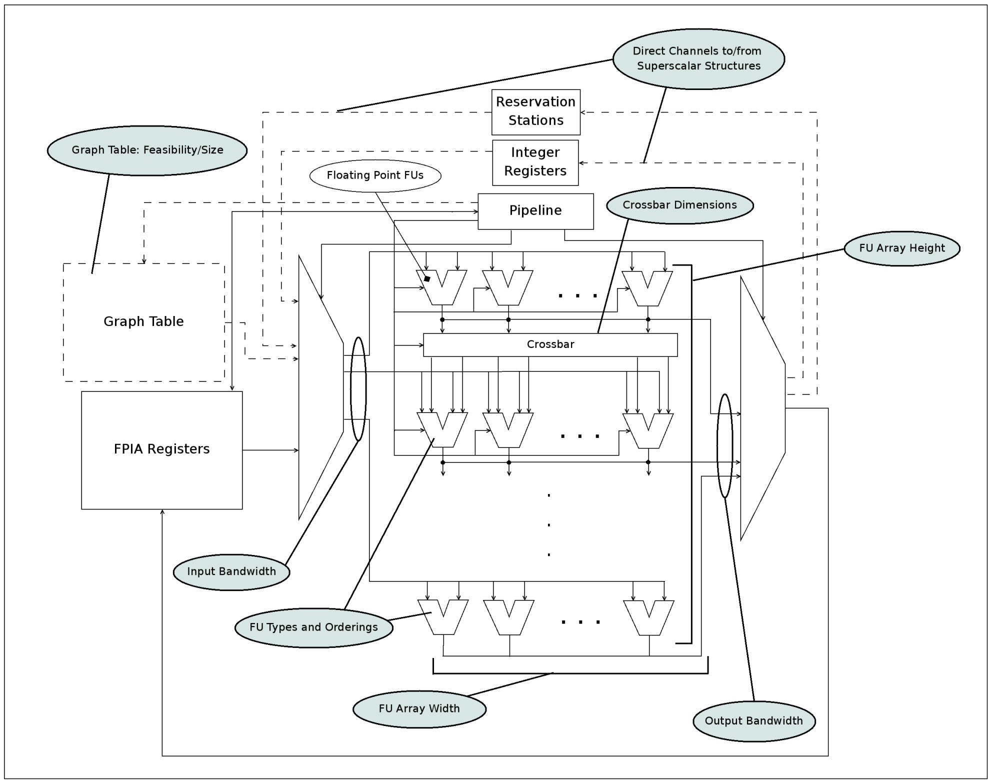


FIG. 2.1. *FPIA Microarchitectural Extensions and Design Parameters*

required register values to and from them. However, the details of such an organization are not immediately clear. For instance, how many FUs should be in the array? What are the dimensions of the array? What interconnect and bandwidth are required? These are just a few of the questions, and the answers are dependent on the characteristics of the FPIAs that are extracted from applications. Hence, it is also important to know the characteristics of FPIAs that exist in applications.

Figure 2.1 shows FPIA extensions to a conventional processor, along with design parameters that need to be specified. The array requires other special hardware shown in the figure. For instance, a special register file contains register values that are fed to and from the FUs in the array. One important question is how values are collected and distributed to and from the conventional register files. Are direct channels provided or are special instructions needed for moving data in and out of the FPIA register file? Another question is where are the FPIAs fetched from? Is it feasible to store them in a small, dedicated memory (shown as the *Graph Table* in the figure), and if so, how large is this table?

These are only a few of the interesting questions, and this work does not seek to answer all of them. The analysis presented in section 5 seeks to characterize the typical FPIA instructions from the extraction process and fill in the design parameters of Figure 2.1 based on these results.

**3. Related Work.** Instruction aggregates have been explored in previous research, but in different forms than considered here. Minigraphs are similar in many aspects to FPIA, but are limited to instruction aggregates with a bounded, small number of inputs and outputs

[2, 1]. Sharkey et al. considered instruction packing, sharing pipeline resources between pairs of instructions, with consideration to performance and power consumption [9]. Macro-op scheduling focuses on combining instructions of single-cycle latency [5]. The issue and execution of instruction groups in instruction level distributed processing is similar to the method considered here, but the organization of hardware, especially that of the FUs, differs [4].

One primary difference between this work and past work is the focus on floating point, for which we must take into account special considerations. Compared to the general case in which integer computations comprise a large portion of the aggregates, the hardware design space is more restricted. Certain details of the capabilities of FUs must be considered, such as the operations that a given FU is able to perform [3]. Run-time reconfiguration considered by work focused on integer computation is not feasible in the context of floating point. Additionally, the interaction of the FPIA components with conventional OoO hardware is an important design consideration. Overall, the specification of hardware design parameters, such as those in Figure 2.1, is of key importance in this context.

**4. Methodology.** The FPIA design and evaluation process begins with the analysis of applications to determine the properties of dependence graphs that can potentially be converted to FPIAs. The simulation of applications enhanced with FPIAs is used to estimate their performance. The analysis tool and simulator are based on the Structural Simulation Toolkit (SST) [6], and their use is centered on eighteen traces of eight applications with varied inputs.

**4.1. Applications.** Traces of a suite of eight of Sandia's scientific applications are used in this study. The applications cover a wide range of purposes, including molecular dynamics (LAMMPS), shock mechanics of solids (CTH), radiation transport (ITS), and circuit simulation (Xyce). These applications have important properties that set them apart from standard benchmarks, such as those found in the SPEC suite. Large basic block sizes, large working sets, and instruction mixes with high percentages of floating point are a few notable properties [7, 8]. In the analysis phase, traces of 4 billion instructions are used for all applications other than sPPM, whose trace is 2 billion instructions. In the simulation phase, a trace of 100 million instructions is used for each application.

**4.2. SST.** For the simulations, we used the Structural Simulation Toolkit (SST) [6]. The frontend of the simulator reads in an instruction trace of the application, and the instructions from the trace are fed to the backend which is based on the SimpleScalar model of an OoO processor. The backend models the activity that occurs to estimate the execution time of an OoO processor. This framework is used unmodified to obtain the baseline performance numbers for comparison against FPIA. The frontend is modified for FPIA code transformation, and the backend OoO model is extended with components for executing the FPIAs. The details of these modifications are given in section 6.1.

**5. Analysis of FPIA Content of Applications.** This section first presents the process used to extract FPIAs from applications. To determine an efficient design based on the parameters given in Figure 2.1, we have gathered statistics to characterize a typical FPIA. The relevant statistics corresponding to the parameters are listed, and the results are presented.

We used a set of traces of Sandia's simulation codes for these experiments. Using traces, as opposed to execution, simplifies the experiments. The goal is to study the FPIA content in the applications. This will justify future work with FPIA extraction with actual codes.

**5.1. FPIA Extraction.** This process for FPIA extraction considers instructions grouped by basic block (instructions occurring between two branches) so that complications arising

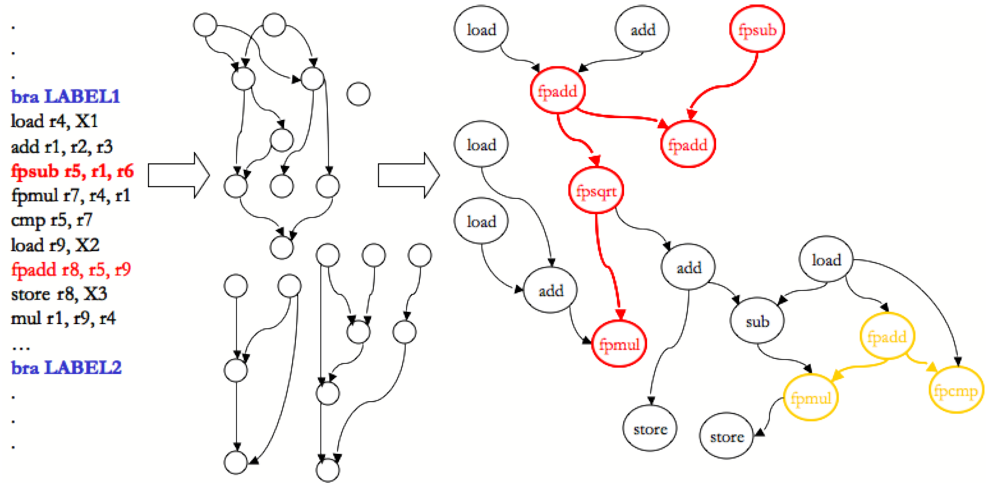


FIG. 5.1. *Overview of FPIA Extraction Process*

due to branch prediction can be ignored. Dependence graphs including all non-FP instructions are first generated for the entire basic block. The subgraphs of FP operations are then extracted from these graphs. This set of subgraphs is used as the basis for the characterization in the next section. Figure 5.1 shows an overview of this process.

Certain complications arise during the extraction process that must be considered. As an example, consider how a graph of FP operations interacts with non-FP operations. It is likely feasible to assume that non-FP instructions providing input register values can be executed before the FPIA, and instructions using register values produced by the FPIA can be executed afterwards. The register values will be propagated to and from the FPIA register file accordingly. However, how should we handle graphs that contain a non-FP instruction that both uses an output from the graph an then produces an input for the graph? In general, this goes back to the question of how the FPIA hardware interacts with the conventional hardware. One option to handle this case is to allow the FPIA FU array to synchronize with the non-FP FUs in order to obtain the needed value. However, this complicates the design. Another option is to cut the graph into pieces so that the intermediate operation is no longer both a consumer and producer for a graph. In these experiments, we have found that this case is rare, occurring in less than 2% of the graphs in most cases, and so we do not aggregate these graphs. However, this is a corner case for FPIA in general that should be considered when necessary.

**5.2. FPIA Characterization.** To determine a set of values for an effective design based on Figure 2.1, we have tracked statistics of FPIAs based on the extraction process given in the previous section. Figure 5.2 shows characteristics of the graphs that correspond to the design parameters. An execution trace of 4 billion instruction is analyzed for each application. All results for these analyses are appended to the end of the paper, and a summary of the results is shown in Figure 5.2. This summary shows that if we design the hardware for a given parameter value, this will allow the execution of a certain percentage of the graphs for sixteen out of eighteen applications, given no other hardware limitations. Considering sixteen applications gives us tolerance for at most two outliers.

The breakdown of this summary is given in Figures A.2 through A.6. These figures show the percentage of graphs we will be able to execute when designing for a given parameter value, given no other limitations. For instance, if we design for graphs of size 16, this will

| Hardware Design Parameter | Graph Property of Interest |
|---|---|
| Graph Table Feasibility/Size | # of Unique Graphs |
| Functional Unit Array Height/Width | Height/Width of Graphs |
| Functional Unit Types/Orderings | Frequency of operation sequences |
| Crossbar Dimensions | Structure of Most Common Graphs |
| Register File Bandwidth | # of Register Inputs/Outputs |
| Memory Bandwidth | # of Loads/Stores to Unique Addrs |

FIG. 5.2. *Map of Parameters to Properties*

| Min Graph Coverage (16/18) | # of ops | Width | Depth | Register Inputs | Register Outputs | Memory Inputs | Memory Outputs | Wide Loads | Wide Stores |
|---|---|---|---|---|---|---|---|---|---|
| 25% | 16 | 1 | 16 | 16 | 16 | 4 | 2 | 2 | 1 |
| 50% | 16 | 1 | 16 | 16 | 16 | 8 | 2 | 4 | 1 |
| 75% | 16 | 1 | 16 | 32 | 16 | 8 | 4 | 8 | 2 |
| 90% | 32 | 2 | 32 | 64 | 16 | 16 | 8 | 16 | 2 |

FIG. 5.3. *Design Parameter Values from Graph Analysis*

allow the execution of about 95% of the graphs in cth.4B.2gas.

Graph size, shown in Figure A.2, is simply the number of vertices in the graphs. Given an ASAP scheduling of the operations in the graph, the width (Figure A.4) is the maximum number of operations in a given layer, and the depth (Figure A.3) is the number of layers. Register inputs (Figure A.5) and outputs (Figure A.6) are the number of unique register values produced and consumed by all instructions in the graph. The number of wide memory inputs (Figure A.7) is an estimate of the number of unique cache lines that are touched with loads, and the number of wide memory outputs (Figure A.8) is an estimate of the number of unique cache lines touched with stores.

The number of unique FP subgraphs in a typical application has implications on the design, such as whether or not storing the FPIAs in a table is feasible, and the size of the table. To determine the number of unique FP subgraphs, we build a list of unique graphs as the application is analyzed. When a new graph is generated, isomorphism is run against each unique graph in the list. In order for two graphs to be isomorphic, they must have the same structure. The results for this analysis is shown in Figure A.1 as *Iso with No Op Comparison*. The analysis was also run for the case in which isomorphism requires that there exist a vertex map matching operations between the graphs. These results are shown in Figure A.1 as *Iso with Op Comparison*. This analysis is performed to take into account that an FP FU may not be able to perform all types of FP operations, which will place additional restrictions on the design space.

**5.3. Conclusions.** From the results of the analyses, we see that the graphs usually have no more than 16 operations, and are narrow and deep. An FU array that has dimensions 16x1 will allow execution of over 75% of the graphs for most applications, while dimensions of 32x2 will allow execution of almost all the graphs. From this, we assume that the typical FPIA we will execute has 32 operations. Using the number of register inputs and outputs from the table, we can get a rough idea of the size of one of these instructions, assuming all information is encoded into a single package. Assuming no more than eight types of FP operations, the operation can be specified in 3 bits. Routing the results among the FU array also requires bits in the instruction. For this approximation, we will assume that each of the 32 operations requires 4 bits to route its results. If we assume 64 entries in the FPIA register file, and design for 32 register inputs, we will need 192 bits for input register values, and then 96 bits for the 16 output register values. As shown in Figure A.1, the number of unique graphs is generally no more than 75 in the case of isomorphism with no op comparison, and no more than 150 with op comparison. But if we are conservative and use the maximum values of 209 and 304, the total sizes of tables for storing all FPIAs, are:

[ 32*3 (ops) + 32*4 (routing) + 192 (inputs) + 96 (outputs) ] bits/inst. = 512 bits/inst.

512 bits/inst. * 209 insts. = 13376 bytes. (no op comparison in isomorphism)

512 bits/inst. * 304 insts. = 19456 bytes. (with op comparison in isomorphism)

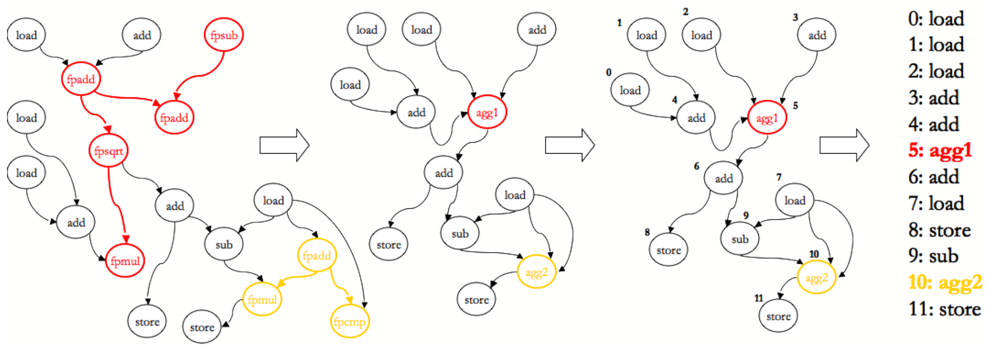
A 16- or 32-KB memory dedicated for storing all FPIAs certainly seems feasible, given that many modern processors have a few MB of on-chip cache. Note that this assumes no immediate values (for now we'll assume that all values come from registers). One other simplifying assumption made by these calculations is that in order for two graphs to be isomorphic, they don't necessarily need the same register indices as inputs and outputs. This factor may change the size of the table, but this consideration is saved for future work.

**6. FPIA Performance Evaluation.** This section explains the approach used for estimating the execution time of an application enhanced with FPIAs run on a machine based on the design shown in Figure 2.1. With the simulation results, we compare this execution time to the time required to execute the original application on a conventional superscalar OoO processor.

**6.1. Simulation Approach.** For this work, we modified the frontend and backend of SST. In the frontend, our extensions extract the dependence graphs from the trace as in the analysis phase described in the previous section. However, in this stage we filter the graphs based on the statistics from the analysis. The FP instructions in a graph are replaced with the FPIA, which is tagged with the appropriate dependencies inherited from the FP instructions.

The backend is extended to include a special FU model for executing FPIAs. Here, we modeled the FU array shown in Figure 2.1 as a single FU object in the simulator. Here are a few important simplifying assumptions made by this model.

First, the simulation ignores structural hazards. If an FPIA has no unresolved dependencies and is issued, it is able to execute. Next, the FU object can execute any FPIA, regardless of dimensions or operation content. Also, since the FU array is modeled as a single FU, we ignore the complexities of scheduling individual operations to FUs. The simulator also ignores any limitation due to instruction encoding, and the delay of an FPIA is assumed to be its critical path with an ASAP scheduling. Finally, output bypassing is implemented, which allows instructions dependent on register values produced by the FPIA to proceed when the register values are available at the outputs of the functional units.



0: load
1: load
2: load
3: add
4: add
**5: agg1**
6: add
7: load
8: store
9: sub
**10: agg2**
11: store

FIG. 6.1. *Overview of FPIA Transformation Process*

**6.2. FPIA Insertion.** One issue that is faced is where the FPIA should be placed in the transformed code. Although the most straightforward approach is to simply choose the position of one of the FPIAs FP instructions, this will lead to problems. Consider four instructions $A$, $B$, $C$, and $D$, appearing in this order in the trace. $B$ is dependent on $A$, and $D$ is dependent on $C$. $A$ and $D$ are also FP instructions that will be replaced by an FPIA, $E$. After $A$ and $D$ are removed, $E$ cannot be inserted in the position of $A$, because it must come after $C$ due to its dependence on $C$. But $E$ cannot be inserted in the position of $D$, because it must come before $B$, since $B$ is dependent on $E$. Since $E$ must come before $B$ but after $C$, the position of $B$ and $C$ must be switched, with $E$ inserted between then.

A solution to the general problem of FPIA insertion is to collapse all FP instructions that will be replaced by an FPIA into a single vertex. Topologically sort the new graph, and order the instructions in order of the sort. This process is shown in Figure 6.1.

Note that after FPIA extraction there will always exist a "good" ordering: one for which all instructions on which instruction $i$ is dependent are before $i$. A topological sort on a dependence graph will always produce a good ordering. So, as long the new graph with the constituent FP operations of the FPIA collapsed into a single vertex is still acyclic, we can get a good ordering through a topological sort. The only way the new graph would contain a cycle is when the FPIA requires an intermediate non-FP operation. This is another advantage of throwing out FPIAs with intermediate non-FP operations. A more thorough proof of the claim is given below.

**Claim.** Given a set $I$ of instructions in a basic block, $\exists$ a good ordering for $I$ after extraction of an FPIA $F$.

**Proof.** To order the instructions, they are inserted into a sequence of slots. It is sufficient to show that if an instruction $i$ is inserted into the next available slot, then $\nexists$ an instruction on which $i$ is dependent that hasn't been inserted (an *available* instruction). This is the same as showing that $\nexists$ a path in the dependence graph from any available instruction to $i$.

All instructions that are part of the FPIA $F$ are in a set $P$. We assume $F$ cannot require intermediate non-FP instructions. This means that $\forall$ pairs of instructions $(x,y) \in P$, $\nexists$ a path from $x$ to $y$ through some $z \notin P$.

Consider each instruction $i \notin P$. If $\exists$ a path from $i$ to some $p \in P$, then $i \in S$. If $\exists$ a path from some $p \in P$ to $i$, then $i \in T$. Otherwise, $i \in U$. $S \cap T = \emptyset$ because if $\exists$ a path from $i$ to some $p \in P$ and $\exists$ a path from some $q \in P$ to $i$, then $\exists$ a path from $q$ to $p$ through $i$, which is a contradiction. $P$, $U$, $S$, and $T$ are disjoint by definition. So, when we consider the instructions in a set, these instructions are distinct from the instructions in the other sets.

Create a subgraph *SG0* consisting of all instructions in $S$. Topologically sort *SG0*. Assuming that the graph was constructed correctly, this implies that in this order, all dependencies for an instruction $i$ from instructions in $S$ will be satisfied beforehand, by definition of topological sort.

We will now show that the instructions in $S$ can be inserted in the first slots by order of the topological sort of *SG0*.

$\forall$ $(s,p)$ | $s \in S$ and $p \in P$, $\nexists$ a path from $p$ to $s$ by definition of $S$. $\forall$ $(s,u)$ | $s \in S$ and $u \in U$, $\nexists$ a path from $u$ to $s$. If a path from $u$ to $s$ exists, then a path from $u$ to some $p \in P$ would exist through $s$, so $u$ would be in $S$ by definition. $\forall$ $(s,t)$ | $s \in S$ and $t \in T$, $\nexists$ a path from $t$ to $s$. If $\exists$ a path from $t$ to $s$, then $\exists$ a path from some $p \in P$ to $t$, a path from $t$ to $s$, and a path from $s$ to some $q \in P$. This implies a path from $p$ to $q$ through $t$ and $s$, which is a contradiction. Hence, the instructions in $S$ can be inserted in the first slots by order of the topological sort of *SG0*.

Schedule $F$ next. All of its dependencies are satisfied, because all of the instructions on which it is dependent are in $S$ and have been inserted. All instructions in $T$ and $U$ can be inserted after the FPIA, because, by definition, $\nexists$ a path from $t \in T$ to $p \in P$ and $\nexists$ a path from $u \in U$ to $q \in P$.

Create a subgraph *SG1* consisting of all instructions in $U$ and $T$. Topologically sort *SG1* and insert the instructions in this order. The topological sort will ensure that all dependencies are satisfied before an instruction in *SG1* is inserted.

**6.3. Results and Conclusions.** The performance results from the simulations are given in Figure 6.2. Each application was run for 100 million instructions.

It is important to note that the reordering of instructions by itself will affect the performance of the application. We are not interested in changes in performance brought about by instruction reordering, but by the FPIAs. The instruction reordering is necessary for correct execution, and in the future we hope to have better insertion algorithms that require fewer changes to the original instruction stream.

We separate the performance benefits of reordering from that of the FPIAs. The leftmost bars in the sets of Figure 6.2 show the speedup of the application with the reordering but with no FPIAs. Then, to calculate speedup for the cases where FPIAs are used, we use either the results before or after reordering as the baseline, whichever is better.

The second bar in the sets, labeled *2-32: Accelerated*, indicate the execution time when all graphs are assigned a delay of 1. The goal of these simulations was to get a rough upper bound on the performance improvement. The next three bars show the performance results with three different graph size limits. That is, for the results labeled *2-32*, only FPIAs within that size range, inclusive, are extracted.

In most cases, the results show that FPIAs are detrimental to performance. However, for one trace, the FPIAs offer a speedup of over 7%. This shows that FPIAs do offer performance benefits in certain situations. Hence, there is a distinction that must be made in future studies between good and bad FPIAs.

There are two primary reasons we believe that FPIAs are detrimental to performance in these simulations. First is that in the current model, all operations in the FPIA must wait on all of the inputs before proceeding. This can cause the FPIA to prevent instructions from being issued that would otherwise be free to execute independently. Since an instruction is prevented from being issued, the instructions that are dependent on it, even those outside the FPIA, will also be stalled, resulting in a potentially large slowdown. To solve this problem, forwarding of input values into the FUs must be modeled so that execution of each operation can proceed when its inputs are ready. The second reason is that the reordering of instructions in several cases results in a penalty that must be incurred on the FPIAs. In these cases, even though the FPIA outperforms the reordered baseline significantly (such as in *lmp.4B.lj.fix* in

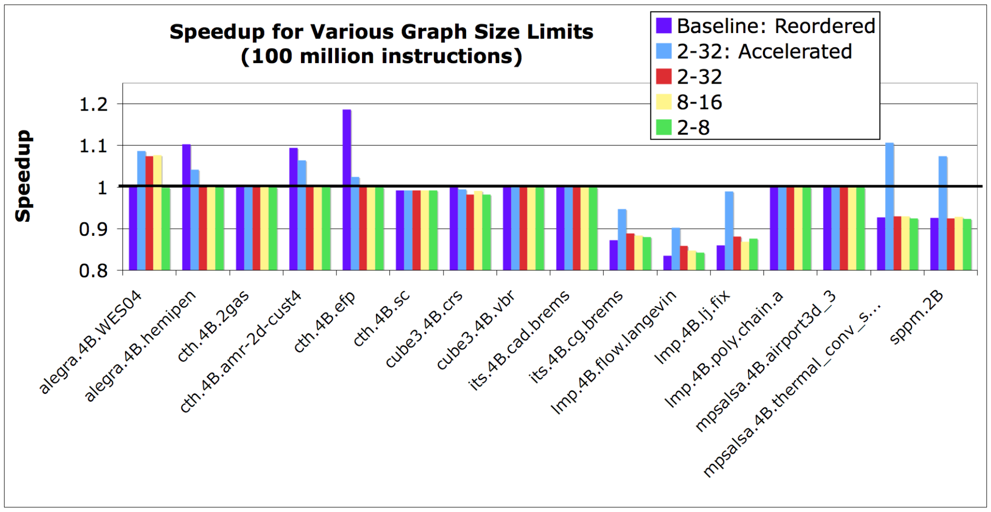


FIG. 6.2. *Performance of Sandia Applications Enhanced with FPIAs of Various Sizes*

Figure 6.2), the improvement is not enough to compensate for the slowdown due to reordering. One approach to dealing with this is to develop a better algorithm for FPIA insertion. Another option is for the extraction tool not to aggregate graphs that will require reordering that leads to lower performance.

**7. Summary and Future Work.** This work has explored the potential of floating point instruction aggregates (FPIA), which seek to lessen the overhead associated with instructions when executed on a superscalar OoO processor. A parameterized microarchitecture design for executing FPIAs was presented, along with a process for extracting FPIAs for execution on this microarchitecture. FPIA content of applications for scientific simulation at Sandia was analyzed to specify values for the parameters of an effective design. A simulator was constructed for estimating the potential performance improvement offered by FPIAs. Although the performance results were not good for most of the applications, we have produced several deliverables that will serve as a basis for future studies:

- Baseline algorithm for FPIA extraction.
- Tools for analyzing FPIA content in applications.
- Methodology and simulator for performance evaluation of FPIAs.

There exist numerous avenues for future work with FPIA. Of primary interest is to determine the properties of FPIAs that are beneficial to performance. While the objective of this work was to determine the properties and potential performance improvement offered by typical FPIA, the next step is to characterize and evaluate *good* FPIAs. The realization that a significant portion of FPIAs actually hurt performance is important. The distinction of *which* FPIAs are good and which are bad will direct the exploration process in the future and guide the design of the toolset and microarchitecture.

Future goals also include production of an improved insertion algorithm, particularly one that interferes less with the work of the compiler. A refined model of the hardware for more accurate performance estimates is also important, with focus on these questions:

- How do FPIA components interact with conventional hardware?
- How do we schedule individual operations to the FUs?
- How do we issue FPIAs in parallel?

The improved model should also implement more efficient feeding of inputs to FPIAs, so that functional units can begin operations as soon as their inputs are available. Finally, the memory requirements of FPIAs are of interest. Is memory performance a bottleneck for FPIAs? Is so, what are effective solutions?

## REFERENCES

[1] A. Bracy, P. Prahlad, and A. Roth, *Dataflow mini-graphs: Amplifying superscalar capacity and bandwidth*, in MICRO 37: Proceedings of the 37th annual IEEE/ACM International Symposium on Microarchitecture, Washington, DC, USA, 2004, IEEE Computer Society, pp. 18–29.

[2] A. Bracy and A. Roth, *Serialization-aware mini-graphs: Performance with fewer resources*, in MICRO 39: Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture, Washington, DC, USA, 2006, IEEE Computer Society, pp. 171–184.

[3] R. M. Jessani and M. Putrino, *Comparison of single- and dual-pass multiply-add fused floating-point units*, IEEE Trans. Comput., 47 (1998), pp. 927–937.

[4] H.-S. Kim and J. E. Smith, *An instruction set and microarchitecture for instruction level distributed processing*, in ISCA '02: Proceedings of the 29th annual international symposium on Computer architecture, Washington, DC, USA, 2002, IEEE Computer Society, pp. 71–81.

[5] I. Kim and M. H. Lipasti, *Macro-op scheduling: Relaxing scheduling loop constraints*, in MICRO 36: Proceedings of the 36th annual IEEE/ACM International Symposium on Microarchitecture, Washington, DC, USA, 2003, IEEE Computer Society, p. 277.

[6] A. Rodrigues, *Programming future architectures: Dusty decks, memory walls, and the speed of light*. Ph.D. Dissertation, University of Notre Dame, 2006.

[7] A. Rodrigues, R. Murphy, P. Kogge, and K. Underwood, *Characterizing a new class of threads in scientific applications for high end supercomputers*, in ICS '04: Proceedings of the 18th annual international conference on Supercomputing, New York, NY, USA, 2004, ACM Press, pp. 164–174.

[8] K. Rupnow, A. Rodrigues, K. Underwood, and K. Compton, *Scientific applications vs. spec-fp: a comparison of program behavior*, in ICS '06: Proceedings of the 20th annual international conference on Supercomputing, New York, NY, USA, 2006, ACM Press, pp. 66–74.

[9] J. J. Sharkey, D. V. Ponomarev, K. Ghose, and O. Ergin, *Instruction packing: Toward fast and energy-efficient instruction scheduling*, ACM Trans. Archit. Code Optim., 3 (2006), pp. 156–181.

## Appendix A. Detailed Results.

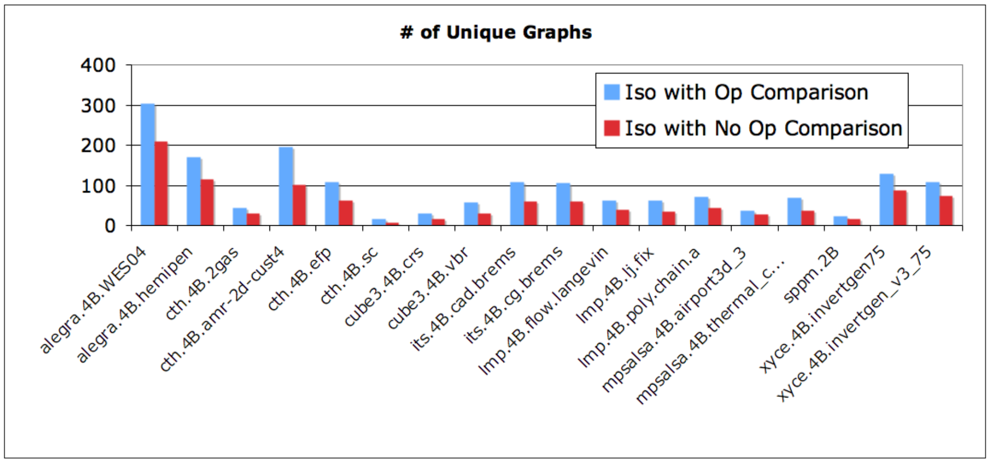


Fig. A.1. *Number of Unique Graphs*

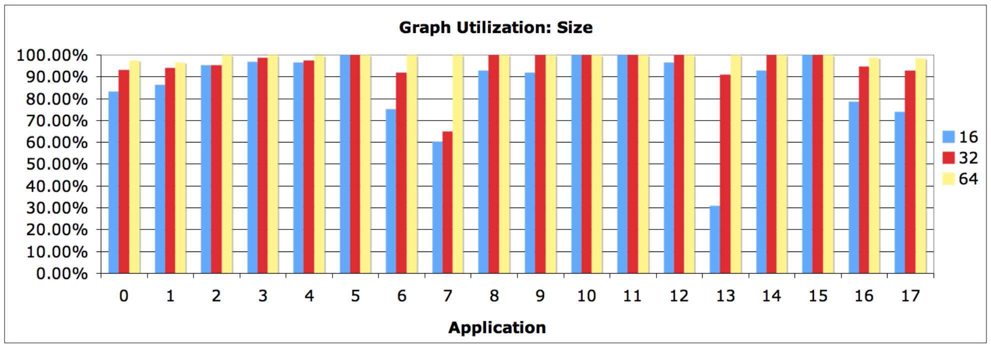


Fig. A.2. *Graph Utilization for Size (applications indexed by order given in Figure A.1).*

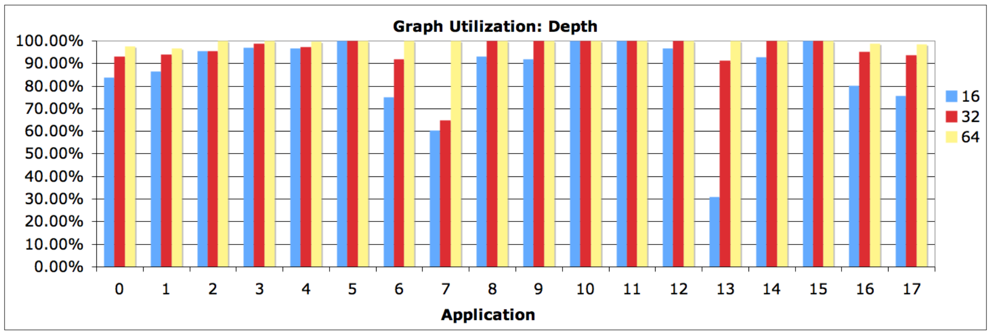


Fig. A.3. *Graph Utilization for Depth*

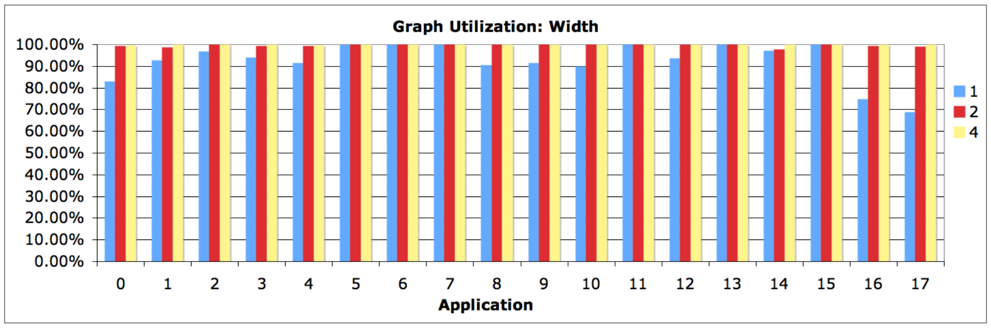


Fig. A.4. *Graph Utilization for Width*

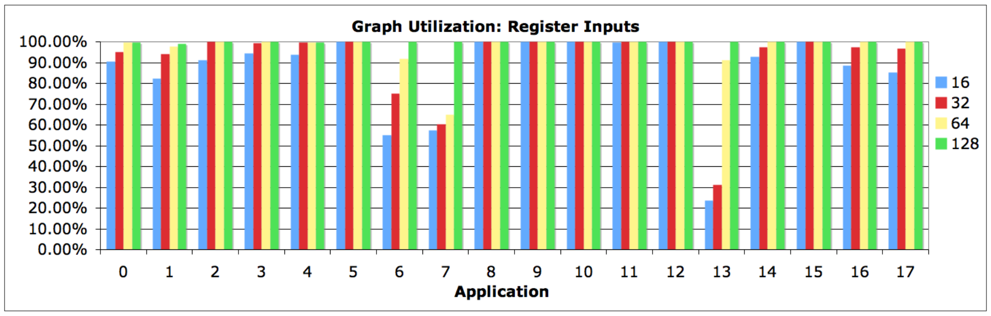


Fig. A.5. *Graph Utilization for Register Inputs*

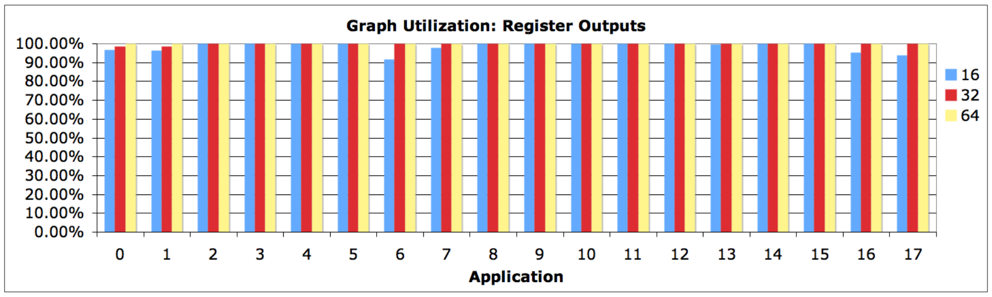


FIG. A.6. *Graph Utilization for Register Outputs*

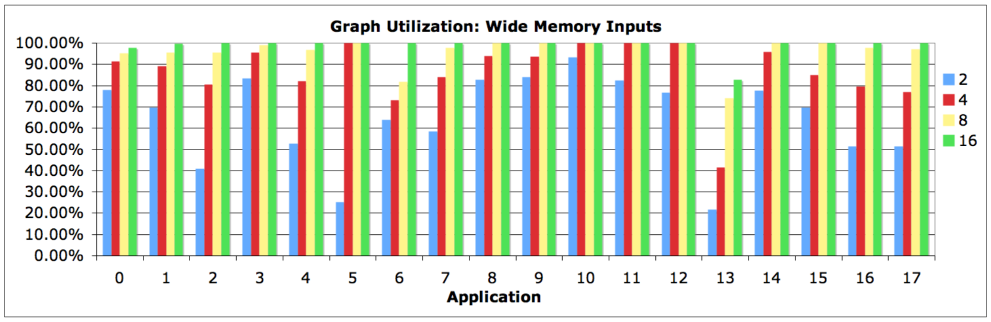


FIG. A.7. *Graph Utilization for Memory Inputs*

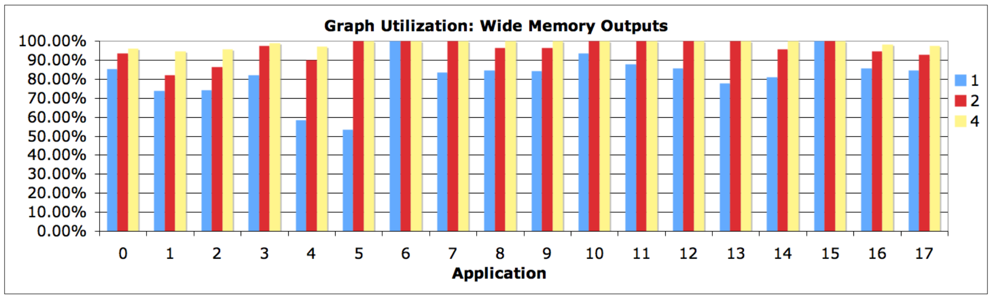


FIG. A.8. *Graph Utilization for Memory Outputs*

# ACCELERATING REED-SOLOMON CODING IN RAID SYSTEMS WITH GPUS

MATTHEW L. CURRY[*], LEE H. WARD[†], ANTHONY SKJELLUM[‡], AND RON B. BRIGHTWELL[§]

**Abstract.** Graphical Processing Units (GPUs) have been applied to many more types of computations than graphics processing for several years. However, until recently, the hardware has not been capable of doing general data processing tasks efficiently. With the advent of more general purpose extensions to GPUs, many more types of computations are now possible than before. One application that we have identified as benefiting from the GPU's unique architecture is Reed-Solomon coding in a manner appropriate for RAID-type systems. In this paper, we show the mapping of the problem to the architecture, implementation concerns, and performance data for such applications.

**1. Introduction.** Graphical Processing Units, also known as GPUs, are massively parallel devices designed to cope with the embarrassingly parallel nature of graphics rendering tasks. Of course, these devices excel at this workload. However, as conventional CPU speedup has stagnated and manufacturers have begun accelerating applications through symmetric multiprocessing on multiple cores, applications developers have been looking toward the GPU as an already mature and highly developed computation platform with dozens of cores. The GPU has been successfully applied to applications that are either embarrassingly parallel or have embarrassingly parallel sub-steps [5, 3].

However, until recently, GPU platforms were restricted in terms of the usable data types. Generally, only certain floating point operations and data types were well supported. Some other types could be emulated through the provided types; however, unless used judiciously, emulation is often an inefficient use of the GPU's resources [6]. With this restriction, only applications that heavily utilized floating point data and calculations could be accelerated via GPU computation.

To solve such problems, and to provide more acceleration for general-purpose GPU (GPGPU) applications, NVIDIA has released its CUDA API and architecture. While CUDA allows for some enhanced hardware features to be accessed, including a very useful local memory, it also allows operations to be performed on arbitrary binary data [7]. This presents the opportunity for applications to perform more general data processing tasks that are well-suited to the GPU's overall architecture, but could not be done well using floating point data alone.

We have identified Reed-Solomon coding as an application that both suits the general architecture of GPUs and requires the more general types and operations available through CUDA. In particular, we believe GPUs would show good performance as part of a RAID-like system that includes more than two checksum disks. This type of calculation is much more computationally complex than those that are performed for one or two checksum disks. In RAID 5, for example, the checksum calculation for byte $n$ can be performed by taking the exclusive-or of byte $n$ on each data disk. Similarly, for two disks, Blaum et. al. have devised an algorithm that uses 50% fewer operations than Reed-Solomon coding does for the same case [2]. However, in the general case, Reed-Solomon coding is the standard method for generating arbitrary amounts of checksum data [4].

In this paper, we describe a system for performing checksum calculations on disk data in the manner of a RAID system with more than two checksum disks. In the second section we give the motivation for systems with more than two checksum disks. In the third section we

[*]University of Alabama at Birmingham, curryml@cis.uab.edu
[†]Sandia National Laboratories, lee@sandia.gov
[‡]University of Alabama at Birmingham, tony@cis.uab.edu
[§]Sandia National Laboratories, rbbrigh@sandia.gov

describe a RAID system that can use a GPU-based checksum component without degrading performance. In the fourth section we describe Reed-Solomon coding implementations on CPUs. In the fifth section we describe how Reed-Solomon coding maps to the GPU. In section six we give the experimental results. In section seven we summarize our findings and describe future work.

**2. Motivation for RAID with Triple-Disk Checksum.** RAID arrays have long been a preferred method of increasing the reliability and speed of secondary storage. There is a broad mix of standard RAID configurations that offer increased performance, data redundancy, or both. For example, RAID 5 configurations can recover from a total failure of one disk while providing increased read and large write performance [4]. RAID 6, which provides the most redundancy of standard RAID levels, is capable of recovering from two simultaneous disk failures. Given the statistics provided by drive manufacturers, these allow for an impressive MTTDL (Mean Time To Data Loss) for a disk array.

However, several researchers have raised concerns about the reliability of disk manufacturers' statistics. Pinheiro et. al. found that the annualized failure rates of their disk drives are much higher than are implied by vendor statistics [10]. Schroeder and Gibson found similar results through a survey of drives under multiple administrative domains [12]. These results imply that RAID reliability cannot be directly calculated from vendors' statistics, but is more likely to be lower.

There are rarer, yet more grave, reliability problems including batch-correlated failures. Batch correlated failures result from a manufacturing defect in a group of drives. Pâris and Long show that if several disks from a defective batch compose a RAID array, chances are poor that one can rebuild a failed disk before *another* disk fails [9]. They also show that adding checksum disks can drastically increase chances of recovery, but even having only two checksum disks can be a risky proposition. Given a recovery time of twenty-four hours and one disk failure per week, a RAID 6 array has less than 70% probability of being able to recover from a disk failure before data loss occurs.

One final issue is double disk failures combined with read errors. Disk drives are approximately doubling in capacity every eighteen months, but disk speeds are not increasing that quickly. Therefore, the time required to rebuild an array is increasing, causing a related increase in the probability of a double disk failure. If all redundancy in the array is removed, there is no protection in the case of a read error from any of the remaining disks; if such an error occurs on one drive, the corresponding data on the other drives is useless, and data is lost. Given the Bit Error Rate (BER) statistics given by drive manufacturers, the probability that a large drive will encounter an unrecoverable read error during the course of reconstruction is too large to ignore.

The authors believe that the most sensible option is to add another disk of checksum information to a RAID array, allowing it to withstand up to three disk failures (total or partial). This is not a commonly implemented solution due to the computational expense over generating checksums for only one or two disks. However, we believe we can utilize the computational power of GPUs as part of a RAID system, allowing three checksum disks while not degrading performance.

**3. A GPU-based RAID System.** In order to create a viable RAID system using GPUs, it is important to address the issue of memory transfers. The GPUs that are typically utilized for computation are outboard devices which operate on their own local memories. Any data that the GPU uses for computation must, at present, be explicitly transferred over the PCI Express bus. In order to keep the disks from falling idle, causing a drop in overall throughput, the system should be able to feed data to disks as quickly as the disks can accommodate.

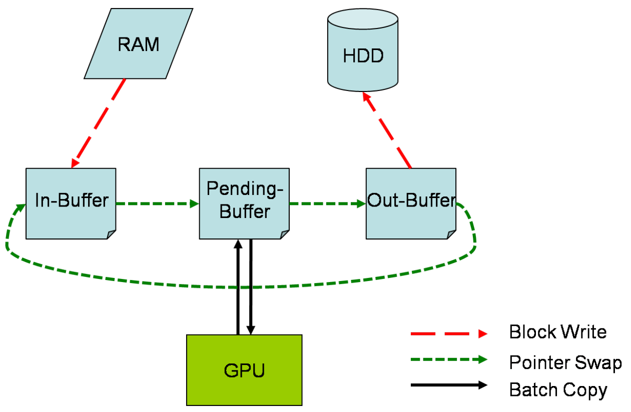


FIG. 3.1. *RAID System Architecture*

One method of doing so is to create a pipelined architecture, as illustrated in Figure 3.1. In this system, we have three buffers: The “in-buffer,” where incoming user write requests are stored; the “pending-buffer,” which acts as the main memory buffer for data being processed by the GPU; and the “out-buffer,” which holds the data and checksum information that is ready to be written to the disks.

As data continually fills what is designated as the in-buffer, these buffers are rotated as both the GPU and hard disks finish with their workloads. This allows the throughput of the system to remain high at the cost of increased latency. Fortunately, in this usage scenario, the user will not notice that her data has not yet been written to disk; in fact, the user may notice that each individual write may complete sooner than expected.

A successful system would show one important characteristic: Given a nearly constant rate of writes, the GPU stage of the pipeline should finish before the disk stage finishes. This would imply that the disks are being utilized to their full abilities, and there would be no throughput decrease due to the computation of the checksums.

**4. Reed-Solomon Coding on CPUs.** In generating arbitrary amounts of redundant data, the primary operation is multiplying part of an information dispersal matrix with a vector of input data elements, yielding another vector of redundant elements satisfying our conditions for RAID checksums. However, rather than relying on integer or floating point arithmetic, the operations are performed on members of a finite field. Addition of two numbers is implemented through an exclusive-or operation, while multiplication by two is implemented through a linear shift feedback register (LSFR). Multiplying two arbitrary numbers involves decomposing the problem into addition of products involving powers of two, which potentially requires a large number of operations. One useful identity which holds true in finite fields of size $2^n$ is:

$$x * y = \texttt{ilog}(\texttt{log}(x) + \texttt{log}(y))$$

where the addition operator denotes normal integer addition modulo $2^n - 1$. Since $n = 8$ for RAID systems, an implementation can contain precalculated tables for the `log` and `ilog` tables, which are 256 bytes apiece. Now, multiplications only require three tables and addition modulo $2^n - 1$ instead of potentially many more logical operations. (While we have given enough of the specifics to continue with discussion, a more in-depth treatment of checksum generation using Reed-Solomon coding is available [11].)

Unfortunately, the type of table lookup operations used in Reed Solomon coding does not exploit the internal vector-based parallelism of CPUs. While fast vector instructions have been included in modern CPUs, few CPU models include a parallel table lookup instruction. In the case of IBM's Power architecture, whose Altivec instruction set includes a parallel table lookup instruction, multiplication in finite fields has been accelerated over typical CPU implementations [1]. Unfortunately, this capability is not common, and CPU implementations tend to suffer due to this lack of capability.

**5. Reed-Solomon Coding on GPUs.** GPUs are architecturally quite different from CPUs. The emphasis of the GPU architecture is to accomplish tens of millions of small, independent, and memory intensive computations per second in order to provide interactive graphics to the user. As such, the GPU has several interesting qualities which are directly applicable to the task of Reed-Solomon coding.

One of the more well-known features of a GPU is its vast number of multithreaded processors. The NVIDIA G80, for example, has been identified by NVIDIA as containing 128 processors. These processors are designed to be effective for small threads of execution on the order of a dozen instructions. In the context of checksum generation, each set of bytes in the data stream (i.e., byte *n* of each data disk) can be viewed as an independent computation. The threading implementation allows for hiding of memory access latency if the ratio of instructions to memory accesses is high, so it is beneficial to pack as many bytes per memory request as possible. In our implementation on the G80, each processor is responsible for sixteen bytes per disk; this is due to the 128-bit memory bus.

Another popular feature of the GPU is its high streaming bandwidth. In our test machine (which will be fully described in our results section), the NVIDIA supplied bandwidth test measures bandwidth between the local and global memory to be over seventy gigabytes per second. In comparison, AMD gives the streaming bandwidth of their Opteron processor to be five gigabytes per second [8]. The checksum computation is streaming in nature; although it is efficient to keep the checksum-in-progress in local memory throughout the computation, the data disk's bytes can be streamed through the GPU one set at a time. The GPU's memory characteristics are more suitable for this type of use.

As an architecture that deals primarily in accessing textures for mapping onto polygons, fast constant accesses are important to good graphics performance. Unlike other types of memory within the CUDA architecture, constant memory is immutable. Therefore, in order to increase performance of constant memory accesses, each processor group's constant accesses are cached. Once data is in this cache, accesses to this data can be as fast as a register access. With this in mind, we have already mentioned a detail of the Reed-Solomon coding algorithm that can make liberal use of this hardware feature: The `log` and `ilog` tables. Each table used in checksum calculations is 256 bytes, while each processor group's constant cache is on the order of many kilobytes.

**6. Results.** We have implemented the GPU and disk components of the system outlined in Figure 3.1. The experiments we have performed are based on the idea that these components will be a part of a full RAID system with three data disks and three checksum disks. As such, the unit of work is phrased as amount of data per disk. The experiment timings include the full "round trip" for the data, in that time begins when data is in main memory of the machine and ends when the computed results finish transferring back to main memory. There is no such transfer overhead to be measured in the CPU case.

The test machine for both the disk and GPU components is as follows: The CPU is an Intel Core 2 Quad 6600. It has four gigabytes of RAM. The disks used are Western Digital Raptor 10,000 RPM SATA hard disks. The GPU is a NVIDIA GeForce 8800 Ultra with 768 MB RAM. The CPU tests were performed on an AMD Opteron 246 processor.

**6.1. Experiment 1: CPU vs. GPU.** For this experiment, we chose James Plank's gflib routines to generate checksums, which uses the same style algorithm as our GPU implementation. While these routines do perform disk I/O, we only timed the checksum generation time of the program. Each program is taking as input three sets of input data, whose individual sizes are indicated on the horizontal axis. Output is three sets of checksums suitable for use in a RAID system that can lose up to three disks.

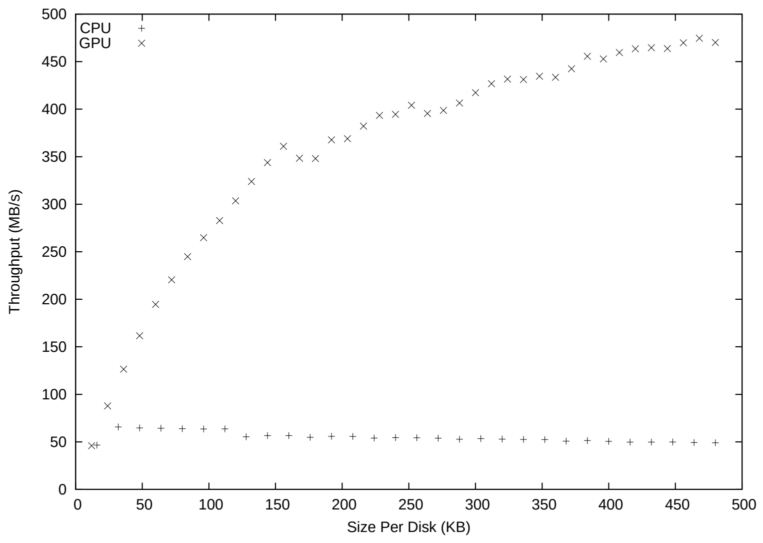


FIG. 6.1. *CPU Checksum Generation vs. GPU Checksum Generation*

As Figure 6.1 indicates, the GPU is much faster than the CPU for all but the smallest payloads. Eventually, for large payloads, the GPU is able to obtain a nine-fold speedup over the CPU.

**6.2. Experiment 2: GPU vs. RAID 0.** This experiment compares the GPU with a three-disk RAID 0 system composed of some of the aforementioned 10,000 RPM SATA disks. It simulates the pipelined system as illustrated in Figure 3.1, allowing us to test our goal of whether the GPU stage of the system pipeline can outperform the disk stage of our pipeline for this case.

The experiment was constructed as follows: For each write size of $n$ kilobytes, many writes of size $3 * n$ kilobytes were generated. This would allow for each data disk to receive $n$ kilobytes each. Depending on the operating mode, these writes were sequentially arranged to allow streaming writes, or the writes were randomly located throughout the disk. The same process was repeated for the GPU, except the idea of a streaming write or random write has no bearing on the GPU's performance, due to the data being prepacked into the buffer.

Referring to Figure 6.2, we see the type of performance we expected. Random write throughput on the disks are much lower than either the GPU's throughput or streaming write throughput. When comparing the streaming write throughput to the GPU throughput, we notice that they exhibit similar performance for small writes. However, the aggregate peak throughput of the disks is much lower than that of the GPU. For large writes, the GPU out-

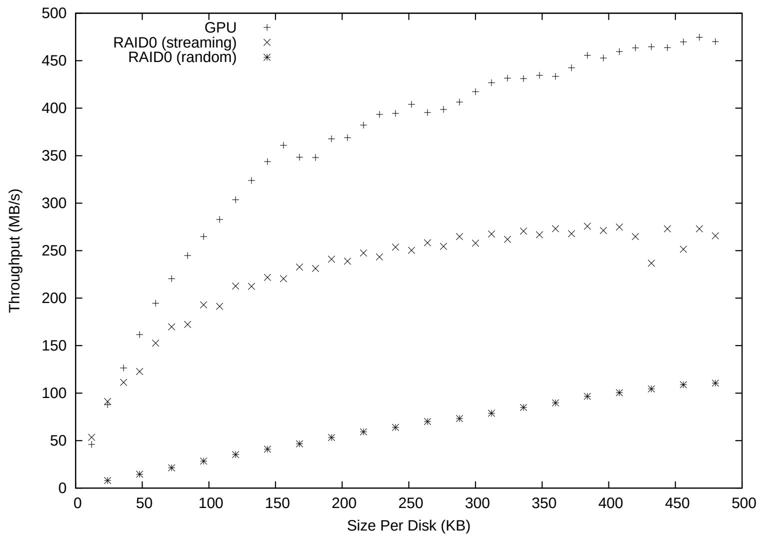


FIG. 6.2. *RAID 0 vs. GPU Checksum Generation*

performs the disks by nearly a factor of two.

**7. Conclusion.** In this paper, we have outlined a GPU-based system for generating redundant information for error recovery in a RAID system that can lose more than two disks. We have benchmarked the system against a CPU implementation, showing a performance improvement of up to nine-fold in some cases. We also benchmarked the throughput of the GPU-based system against a RAID 0 configuration of disks, showing that the GPU system would not be a bottleneck in a system involving six disks, three of which contain redundant data.

In the future, we hope to integrate this into a true RAID system with more data disks. We also plan to outline the performance changes as GPU architectures and interconnects evolve. One interesting possible event is the combining of the GPU and the CPU, which has long been rumored in the industry and has been further fueled by the ATI acquisition by AMD. Such a development would have large implications on a system like this due to the high percentage of time spent transferring data between the GPU and main memory.

## REFERENCES

[1] R. Bhaskar, P. K. Dubey, V. Kumar, and A. Rudra, *Efficient Galois field arithmetic on SIMD architectures*, in SPAA '03: Proceedings of the fifteenth annual ACM symposium on Parallel algorithms and architectures, New York, NY, USA, 2003, ACM Press, pp. 256–257.

[2] M. Blaum, J. Brady, J. Bruck, and J. Menon, *EVENODD: an optimal scheme for tolerating double disk failures in RAID architectures*, in Proceedings the 21st Annual International Symposium on Computer Architecture, 1994, pp. 245–254.

[3] N. A. Carr, J. Hoberock, K. Crane, and J. C. Hart, *Fast GPU ray tracing of dynamic meshes using geometry images*, in GI '06: Proceedings of Graphics Interface 2006, Toronto, Ont., Canada, Canada, 2006, Canadian Information Processing Society, pp. 203–209.

[4] P. M. Chen, E. K. Lee, G. A. Gibson, R. H. Katz, and D. A. Patterson, *RAID: High-performance, reliable secondary storage*, ACM Computing Surveys, 26 (1994), pp. 145–185.

[5] N. Galoppo, N. K. Govindaraju, M. Henson, and D. Manocha, *LU-GPU: Efficient algorithms for solving dense linear systems on graphics hardware*, in SC '05: Proceedings of the 2005 ACM/IEEE conference on Supercomputing, Washington, DC, USA, 2005, IEEE Computer Society, p. 3.

[6] D. Göddeke, R. Strzodka, and S. Turek, *Accelerating double precision FEM simulations with GPUs*, in Proceedings of the 18th Symposium on Simulation Technique (ASIM 2005), F. Hülsemann, M. Kowarschik, and U. Rüde, eds., SCS Publishing House e.V, Sept. 2005, pp. 139–144.

[7] NVIDIA, *NVIDIA CUDA compute unified device architecture programming guide*, 2007.

[8] D. O'Flaherty and M. Goddard, *AMD Opteron processor benchmarking for clustered systems*, 2003.

[9] J.-F. Pâris and D. D. E. Long, *Using device diversity to protect data against batch-correlated disk failures*, in StorageSS '06: Proceedings of the second ACM workshop on Storage security and survivability, New York, NY, USA, 2006, ACM Press, pp. 47–52.

[10] E. Pinheiro, W.-D. Weber, and L. A. Barroso, *Failure trends in a large disk drive population*, in FAST'07: Proceedings of the 5th conference on USENIX Conference on File and Storage Technologies, Berkeley, CA, USA, 2007, USENIX Association, pp. 2–2.

[11] J. S. Plank, *A tutorial on Reed-Solomon coding for fault-tolerance in RAID-like systems*, Software – Practice & Experience, 27 (1997), pp. 995–1012.

[12] B. Schroeder and G. A. Gibson, *Disk failures in the real world: what does an MTTF of 1,000,000 hours mean to you?*, in FAST'07: Proceedings of the 5th conference on USENIX Conference on File and Storage Technologies, Berkeley, CA, USA, 2007, USENIX Association, pp. 1–1.

## Applications

Necessity is the mother of invention and, ultimately, applications drive the advances in computational science, mathematics, and algorithms. The papers in this section span several disciplines, and utilize advanced mathematical and computational tools to address specific problems and applications in their respective fields. In particular, each paper utilizes and extends the capabilities of one or more production Sandia codes, including Trilinos (multipackage software framework for large-scale scientific problems), LAMMPS (massively parallel molecular dynamics), Sacado (automatic differentiation), ALEGRA (magnetohydrodynamics code), Intrepid (compatible discretization library), Tramonto (molecular density functional theory), Sundance (rapid development environment for parallel finite element methods), and LOCA (library of continuation methods).

Schiemenz and Robinson present a new method for calculating the Lorentz force in ALEGRA magnetohydrodynamics modeling. They show that deriving the forces directly from a magnetic energy functional has distinct advantages in terms of discrete energy conservation. Their method utilizes the Intrepid compatible discretization package and the Sacado automatic differentiation package, both within the Trilinos framework. Oguntade and Plimpton studied the effects of shape on the behavior of colloidal suspensions utilizing recent additions to the LAMMPS molecular dynamics package. The effect of shape on the rheology of suspensions is a largely unexplored area of colloidal research. Specifically, the rotational and translational diffusivities of aspherical particles in solvents were measured. Knepper and Heroux consider Schur complement techniques to reduce the run-time and memory requirements for solution of equations arising from the use of density functional theories for inhomogeneous fluids. They and also consider how the addition of Coulomb effects changes the problem structure for the Schur complement approach. Fettig and Collis developed a parallel simulation tool for modeling microvascular self-healing flows using Sundance. They simulate and analyze the flow of a solidifying fluid in a crack as it would occur in proposed self-healing composites. In particular, their simulator is capable of modeling free surface flows, dynamic contact lines, and polymerizing flow in small scales. Dickson *et al.* consider a recently proposed theory describing structural and thermodynamic properties of atomic and molecular fluids. They reformulate the new theory into a form conducive to performing a continuation study in density for atomic fluids, and present continuation results for atomic fluids using the Trilinos software package LOCA. Chowdhary and Robinson discuss the derivation and implementation of a fast and accurate method for the evaluation of the magnetic vector potential associated with a circular loop current source field. They have implemented this method in the ALEGRA code to allow for 2D and 3D circular loop source fields for magnetic diffusion simulations.

M.L. Parks
S.S. Collis

December 6, 2070

# ENERGY BASED MAGNETIC FORCE COMPUTATION USING AUTOMATIC DIFFERENTIATION

ALAN R. SCHIEMENZ[*] AND ALLEN C. ROBINSON[†]

**Abstract.** A new method for magnetic energy force discretization for ALEGRA magnetohydrodynamics modeling is considered. The method combines the automatic differentiation technology from the Trilinos Sacado project with the Intrepid compatible discretization library to compute the magnetic forces. Deriving the forces directly from a magnetic energy functional is shown to have distinct advantages in terms of discrete energy conservation properties. Reducing the cost of this computation remains a challenge.

**1. Introduction.** Arbitrary Lagrangian-Eulerian (ALE) modeling is an important methodology for computing interaction of materials at high stresses and energies. Lagrangian modeling implies that the mesh description and representation moves with the underlying material. Mass, momentum and energy conservation are modeled in a Lagrangian frame of reference. Lagrangian analysis may be useful for following material motion with a carefully designed mesh to track desired features. However, in many cases the material motion will eventually cause meshes to tangle and the mesh must be smoothed and straightened and the variables remapped. If the new mesh is exactly the same as the old mesh then this is the so-called Eulerian limit for an ALE code. When the physics of the problem involves high current and magnetic fields then we must add a reduced form of Maxwell's equations that includes Faraday's law in moving media and Ampere's law in which the displacement current has been neglected. The resulting combined equations are known under the general title of magneto-hydrodynamics (MHD). Energy can be exchanged between the magnetic field and material kinetic energy through the Lorentz or $\mathbf{J} \times \mathbf{B}$ forces. If the material has an infinite conductivity then magnetic flux through a moving surface is invariant and energy is exchanged directly between the magnetic field and the material kinetic energy. Finite conductivity will result in an update to the magnetic flux through a magnetic diffusion equation which dissipates energy through Joule heating. An important example of the need for MHD modeling is with the Z-machine at Sandia National Laboratories. MHD modeling is used to design record breaking magnetically driven quasi-isentropic flyers for equation of state experiments and to study the physics of wire-array implosions [4]. Key to predictive results with these computations is to properly account for the energy in the problem and to ensure that the actual physical partitioning of energy between magnetic, kinetic and internal energy is correctly modeled. Thus, discrete measures of the energy are important and each aspect of the numerical modeling methodology needs to be examined for its effect on the partitioning. ALEGRA is an MHD ALE code developed at Sandia National Laboratories that is used to analyze these high energy physical systems [7]. We discuss in this paper a method for computing the Lorentz force in ALEGRA which is consistent with the discrete form of the energy used for the magnetic diffusion.

**2. Ideal Magnetohydrodynamics.** The ALE sequence for modeling MHD in ALEGRA in both 3D and 2D planar fields consists of an operator split in the Lagrangian motion between an ideal MHD step and a magnetic diffusion step. First, an ideal MHD step is performed in which nodal forces and accelerations are computed and nodes are moved accordingly while holding magnetic fluxes or magnetic vector potential circulations invariant. Secondly, an implicit magnetic diffusion step may occur, utilizing the eddy current approximation to Maxwell's equations. Finally, an optional remeshing and remapping step may be

[*]Brown University, ars@dam.brown.edu
[†]Sandia National Laboratories, acrobin@sandia.gov

taken. The details of this remesh/remap step are beyond the scope of this paper and we are concerned primarily with the force description in the ideal MHD step.

Of particular interest is the first of these steps, where nodal forces are calculated. One way of accomplishing this is to approximate $\mathbf{J} \times \mathbf{B}$ in some way and from this compute the electromagnetic force at the nodes. Alternatively, one can utilize the Maxwell stress tensor to compute the Lorentz force,

$$\mathbf{J} \times \mathbf{B} = \nabla \cdot \mathbf{T}^M - \mathbf{B}(\nabla \cdot \mathbf{B}) \tag{2.1}$$

$$= \nabla \cdot \mathbf{T}^M \tag{2.2}$$

since $\nabla \cdot \mathbf{B} = 0$. The components of $\mathbf{T}^M$ are given as

$$\mathbf{T}^M = \frac{1}{\mu}\left(\mathbf{B} \otimes \mathbf{B} - \frac{1}{2}\mathbf{B}^2\mathbf{I}\right), \tag{2.3}$$

or in component form,

$$\mathbf{T}^M_{ij} = \frac{1}{\mu}\left(B_i B_j - \frac{1}{2}\delta_{ij} B_k B_k\right). \tag{2.4}$$

In ALEGRA the magnetic stress tensor is approximated by evaluating $\mathbf{T}^M$ at element centers. This stress can then be fed to the finite element divergence operator to compute the nodal force. There is no built-in mechanism, however, in this approach to enforce any particular discrete correspondence between the forces and the magnetic energy change. We expect consistency only to within truncation errors. We propose an alternative approach that utilizes what is known about the physics of the problem and will help maintain a discrete energy conservation principle which is exact in the limit of small differential motions. This idea has been utilized previously for MHD code development and may be referred to as the principle of virtual work [6]. We derive below why the principle of virtual work is useful and how it can be derived from the continuum equations. The books by Eringen and Maugin and Moreau are useful references for the mathematics of electrodynamics and MHD [2, 5].

Ideal MHD adds the magnetic induction equation to the mass momentum and energy conservation equation. In integral form the induction equation appears as

$$\frac{d}{dt}\int_{S_t} \mathbf{B} \cdot \mathbf{n} dA + \int_{\delta S_t} \mathbf{E}' \cdot \mathbf{ds} = 0 \tag{2.5}$$

where $S_t$ is any moving surface and $\mathbf{E}'$ is the co-moving electric field integrated around the boundary of the surface. The surface is a material surface so that taking the time derivative inside the integral and insisting that this equation hold for all surfaces results in

$$\frac{\partial \mathbf{B}}{\partial t} + \nabla \times (\mathbf{B} \times \mathbf{v}) + \mathbf{v}(\nabla \cdot \mathbf{B}) + \nabla \times \mathbf{E}' = 0. \tag{2.6}$$

Since we require the condition $\nabla \cdot \mathbf{B} = 0$ we have the familiar relation

$$\frac{\partial \mathbf{B}}{\partial t} + \nabla \times \mathbf{E} = 0 \tag{2.7}$$

where

$$\mathbf{E} = \mathbf{E}' + \mathbf{B} \times \mathbf{v}. \tag{2.8}$$

Note that this is the familiar relationship for Ohm's law in the laboratory frame of reference,

$$\mathbf{J} = \sigma\mathbf{E}' = \sigma(\mathbf{E} + \mathbf{v} \times \mathbf{B}). \tag{2.9}$$

For ideal MHD, $\mathbf{E}' = 0$, so that the induction equation becomes

$$\frac{\partial \mathbf{B}}{\partial t} + \nabla \times (\mathbf{B} \times \mathbf{v}) = 0. \tag{2.10}$$

By multiplying this equation by the magnetic field $\mathbf{H} = \mathbf{B}/\mu$ and integrating over a moving volume we obtain the electromagnetic power integral as follows:

$$\frac{\partial}{\partial t}\frac{\mathbf{B}^2}{2\mu} + \frac{\mathbf{B}}{\mu} \cdot \nabla \times (\mathbf{B} \times \mathbf{v}) = 0 \tag{2.11}$$

$$\frac{\partial}{\partial t}\left(\frac{\mathbf{B}^2}{2\mu}\right) + \nabla \cdot (\frac{\mathbf{B}^2}{2\mu} \cdot \mathbf{v}) + \nabla \cdot ((\mathbf{B} \times \mathbf{v}) \times \mathbf{H} - \frac{\mathbf{B}^2}{2\mu} \cdot \mathbf{v}) + (\mathbf{B} \times \mathbf{v}) \cdot \mathbf{J} = 0 \tag{2.12}$$

$$\frac{\partial}{\partial t}\left(\frac{\mathbf{B}^2}{2\mu}\right) + \nabla \cdot (\frac{\mathbf{B}^2}{2\mu} \cdot \mathbf{v}) - \nabla \cdot (\mathbf{T}^M \cdot \mathbf{v}) + (\mathbf{B} \times \mathbf{v}) \cdot \mathbf{J} = 0 \tag{2.13}$$

$$\frac{d}{dt}\int_{V_t} \frac{\mathbf{B}^2}{2\mu}\, dV = -\int_{V_t} (\mathbf{J} \times \mathbf{B}) \cdot \mathbf{v}\, dV + \int_{\delta V_t} \mathbf{T}^M \mathbf{n} \cdot \mathbf{v}\, dA \tag{2.14}$$

Note that $\mathbf{J} = \nabla \times \mathbf{H}$ from the reduced form of Faraday's law. Similarly, there is a power expended theorem for the momentum equation [3]

$$\frac{d}{dt}\int_{V_t} \frac{\mathbf{v}^2}{2}\rho\, dV = \int_{V_t} (\mathbf{J} \times \mathbf{B}) \cdot \mathbf{v}\, dV + \int_{V_t} \mathbf{T} \cdot \mathbf{D}\, dV + \int_{\delta V_t} \mathbf{T}\mathbf{n} \cdot \mathbf{v}\, dA \tag{2.15}$$

where $\mathbf{D}$ is the symmetric part of the velocity gradient tensor and $\mathbf{T}$ is the thermodynamic Cauchy stress. Therefore if we add these two last equations we obtain

$$\frac{d}{dt}\int_{V_t} \left(\frac{\mathbf{B}^2}{2\mu} + \rho\frac{\mathbf{v}^2}{2}\right) dV = \int_{\delta V_t} \mathbf{T}^M \mathbf{n} \cdot \mathbf{v}\, dA + \int_{V_t} \mathbf{T} \cdot \mathbf{D}\, dV + \int_{\delta V_t} \mathbf{T}\mathbf{n} \cdot \mathbf{v}\, dA \tag{2.16}$$

which indicates that the sum of material derivative of the kinetic and magnetic energy in a moving volume is independent of the magnetic force power $\int_{V_t} (\mathbf{J} \times \mathbf{B}) \cdot \mathbf{v}\, dV$. Fundamentally this means that there is a direct exchange between the magnetic and kinetic energy in ideal MHD which must hold independently of any thermodynamic stresses. It is this principle that allows us to compute a formula for the relationship of the change in magnetic energy to a material force.

Consider a Lagrangian finite element with coordinates $\{\mathbf{x}_k\}_{k=1}^N$, where $\mathbf{x}_k = (x_k, y_k, z_k)$. A hexahedron, for example, is defined by its $N = 8$ vertices, yielding $8 \times 3 = 24$ coordinates. For zero thermodynamic stress and boundary velocity, energy conservation implies

$$\frac{d}{dt}\int_{V_t} \left(\frac{\mathbf{B}^2}{2\mu} + \rho\frac{\mathbf{v}^2}{2}\right) dV = 0 \tag{2.17}$$

or

$$\frac{d}{dt}\left[E_K + E_M\right] = 0 \tag{2.18}$$

where $E_K$ refers to the kinetic energy and $E_M$ refers to the magnetic (potential) energy. For a Lagrangian model the mass on a node is invariant and for ideal MHD the flux through a face is constant as in equation 2.5. Therefore,

$$\frac{d}{dt}\left[\sum_k \left\{\frac{1}{2}mv_{x_k}^2 + \frac{1}{2}mv_{y_k}^2 + \frac{1}{2}mv_{z_k}^2 +\right\} + E_M(\mathbf{x}_1, ..., \mathbf{x}_N, \mathbf{\Phi})\right] = 0 \tag{2.19}$$

where $m$ refers to mass and $v_{x_k}, v_{y_k}, v_{z_k}$ refers to velocities in the $x_k$,$y_k$ and $z_k$ directions, respectively, and $\mathbf{\Phi}$ is a representation of the magnetic flux degrees of freedom on each face. Applying the chain rule we have

$$\begin{aligned}\sum_k \left(mv_{x_k}a_{x_k} + \frac{\partial E_M}{\partial x_k}v_{x_k}\right) &= 0\\ \sum_k \left(mv_{y_k}a_{y_k} + \frac{\partial E_M}{\partial y_k}v_{y_k}\right) &= 0\\ \sum_k \left(mv_{z_k}a_{z_k} + \frac{\partial E_M}{\partial z_k}v_{z_k}\right) &= 0\end{aligned} \tag{2.20}$$

where $\mathbf{a}_k = (a_{x_k}, a_{y_k}, a_{z_k})$ are the accelerations in each direction. To enforce these equations we require each relationship hold for arbitrary velocity, yielding

$$\mathbf{F}_k + \frac{\partial E_M}{\partial \mathbf{x}_k} = 0 \;\; \forall k \tag{2.21}$$

where $\mathbf{F}_k = (F_{x_k}, F_{y_k}, F_{z_k}) = m\mathbf{a}_k$ is the desired nodal force. Therefore we have

$$\mathbf{F}_k = -\frac{\partial E_M}{\partial \mathbf{x}_k} \tag{2.22}$$

$$= \frac{\partial}{\partial \mathbf{x}_k}\left(-\frac{1}{2\mu}\int \mathbf{B}\cdot\mathbf{B}d\mathbf{x}\right). \tag{2.23}$$

Thus, differentiating the magnetic energy functional with respect to coordinates will give a computation of nodal force which exactly matches the physics of the problem.

Unlike the method in which the Maxwell stress tensor $\mathbf{T}^M$ is evaluated at cell centers and then differenced via the finite element divergence operator to obtain forces at the nodes, this method will calculate the force in a way that is consistent with the underlying physics and will lead to an exact correspondence between the change in magnetic and kinetic energy with respect to infinitesimal perturbations. However, to do this there are two important factors that must be considered. First, we must be able to exactly compute the magnetic energy in a discrete sense. As seen in equation (2.23), this can be viewed as a functional of the **B**-field. Second, we must exactly compute the derivative of this functional with respect to the coordinates.

The first of these requirements is carried out by the on-going Sandia Intrepid project, a templated compatible discretization technology. The second is provided by Sandia's Trilinos Sacado project, a C++ automatic differentiation library. Due to Intrepid's use of C++ templated data types, the two projects are easily combined to provide an effective means of differentiating the energy functional with respect to element coordinates. This paper examines the utility of this approach for managing the energy exchange between the discrete magnetic and discrete kinetic energy. The discrete magnetic energy functional that we have in mind is exactly the same functional that is used in the finite element solution of the magnetic diffusion or eddy current equation when $\mathbf{E}'$ is non-zero. This functional will depend the finite element solution methodology and in particular will depend on the finite element quadrature rule.

**3. Compatible Spatial Discretizations.** In our three-dimensional ideal MHD setting we have invariant magnetic fluxes (or magnetic potential circulations) throughout the Lagrangian step. That is, the degrees of freedom we work with are fluxes

$$\Phi_i = \int_{\mathcal{F}_i} \mathbf{B} \cdot d\mathbf{a}, \tag{3.1}$$

where $\mathcal{F}_i$ denotes face $i$ of an element $K$. Accordingly, we can express the magnetic flux density as

$$\mathbf{B} = \sum_i \Phi_i \mathbf{W}_i(\mathbf{x}) \tag{3.2}$$

given a corresponding set of (vector) basis functions $\{\mathbf{W}_i\}$.

These basis functions induce a bilinear form corresponding to a mass matrix $\mathbf{M}$, given as

$$M_{ij} = \int_K \mathbf{W}_i \cdot \mathbf{W}_j d\mathbf{x}. \tag{3.3}$$

The particulars of how to compute this mass matrix, e.g. exactly how the bilinear form is evaluated, are handled in Intrepid and are not of concern from the ALEGRA application point of view. In particular, it is not even required that the underlying bilinear form implementation be finite element based. Given this mass matrix, we may now compute the magnetic energy functional:

$$\begin{aligned} E &= \frac{1}{2\mu} \int_K \mathbf{B} \cdot \mathbf{B} d\mathbf{x} && (3.4)\\ &= \frac{1}{2\mu} \int_K \left( \sum_i \Phi_i \mathbf{W}_i \right) \cdot \left( \sum_j \Phi_j \mathbf{W}_j \right) d\mathbf{x} && (3.5)\\ &= \frac{1}{2\mu} \sum_{i,j} \Phi_i \left( \int_K M_{ij} d\mathbf{x} \right) \Phi_j && (3.6)\\ &= \frac{1}{2\mu} \mathbf{\Phi}^T \mathbf{M} \mathbf{\Phi}. && (3.7) \end{aligned}$$

Recalling equation (2.22), computation of nodal force contributions now requires evaluating the derivative of this functional with respect to the coordinates $\mathbf{x}_k$. As the fluxes $\mathbf{\Phi}$ are invariant in the Lagrangian step, only the matrix $\mathbf{M}$ requires differentiation

$$\mathbf{F}_k = -\frac{\partial E}{\partial \mathbf{x}_k} = -\frac{1}{2\mu} \mathbf{\Phi}^T \left( \frac{\partial \mathbf{M}}{\partial \mathbf{x}_k} \right) \mathbf{\Phi}. \tag{3.8}$$

Because a single node may be shared by multiple elements, the total force on a node is the sum of the contributions from each element.

**3.1. Two-dimensional implementation.** A similar approach may be followed for two-dimensional problems, with some slight modifications. In three dimensions, the motivation for representing $\mathbf{B}$ in terms of the magnetic flux degrees of freedom as in equation (3.2) came from the fact that these fluxes remained constant under Lagrangian motion in the ideal MHD step. It is equivalent to require that the vector potential circulations on element edges remain invariant. In ALEGRA the degrees of freedom for planar two-dimensional problems with field in the plane, however, are the out-of-plane magnetic vector potential components. These components are invariant under Lagrangian motion in ideal MHD.

Consider the curl as an operator that acts on vector fields normal to the plane,

$$\mathbf{A}(x, y) = \gamma(x, y)\mathbf{k}. \tag{3.9}$$

Then, the curl operator will map scalar functions $\gamma$ into the $(\mathbf{i}, \mathbf{j})$ plane

$$\nabla \times \mathbf{A} = \mathbf{i}\frac{\partial \gamma}{\partial y} - \mathbf{j}\frac{\partial \gamma}{\partial x}. \tag{3.10}$$

The invariant degrees of freedom $\gamma_l = \gamma(x_l, y_l)$ are the values of the (scalar) potential at the vertices of the element

$$\mathbf{A}(x, y) = \sum_l \gamma_l W_l(x, y)\mathbf{k}, \tag{3.11}$$

where now the basis functions $\{W_l\}$ are scalar-valued. The magnetic energy functional on an element $K$ is then computed as

$$\begin{aligned}
E &= \frac{1}{2\mu}\int_K (\nabla \times \mathbf{A}) \cdot (\nabla \times \mathbf{A})\, d\mathbf{x} && (3.12)\\
&= \frac{1}{2\mu}\int_K \left[\sum_k \gamma_k \left(\mathbf{i}\frac{\partial W_k}{\partial y} - \mathbf{j}\frac{\partial W_k}{\partial x}\right)\right] \cdot \left[\sum_l \gamma_l \left(\mathbf{i}\frac{\partial W_l}{\partial y} - \mathbf{j}\frac{\partial W_l}{\partial x}\right)\right] d\mathbf{x} && (3.13)\\
&= \frac{1}{2\mu}\sum_{k,l} \gamma_k \left(\int_K \frac{\partial W_k}{\partial x}\frac{\partial W_l}{\partial x} + \frac{\partial W_k}{\partial y}\frac{\partial W_l}{\partial y} d\mathbf{x}\right)\gamma_l && (3.14)\\
&= \frac{1}{2\mu}\boldsymbol{\Gamma}^T \mathbf{S}\boldsymbol{\Gamma} && (3.15)
\end{aligned}$$

where the entries of the stiffness matrix $\mathbf{S}$ are defined as

$$S_{ij} = \int_K \nabla W_i \cdot \nabla W_j d\mathbf{x}. \tag{3.16}$$

Finally, as $\boldsymbol{\Gamma}$ remains constant in the ideal MHD step, the nodal force contribution is

$$\mathbf{F}_k = -\frac{\partial E}{\partial \mathbf{x}_k} = -\frac{1}{2\mu}\boldsymbol{\Gamma}^T \left(\frac{\partial \mathbf{S}}{\partial \mathbf{x}_k}\right)\boldsymbol{\Gamma}. \tag{3.17}$$

**4. Automatic Differentiation.** The Intrepid project gives the needed machinery to compute the magnetic energy functional, but we must differentiate this functional with respect to the element coordinates to evaluate the nodal force contributions. For hexahedral grids this differentiation is a non-trivial calculation due to the Jacobian factors in the integrals defining the entries of $\mathbf{M}$ and $\mathbf{S}$. These factors arise in the nonlinear transformation between physical and reference space [1]. We automate the evaluation of these derivatives by utilizing automatic differentiation technology from the Sandia National Laboratories Trilinos Sacado project.

The premise of automatic differentiation is to break down all complicated functions into compositions of elementary functions which can be easily evaluated and differentiated. Then,

Table 4.1
*Example of automatic differentiation for the evaluation of the function* $z(x,y) = x^2 \sin(e^x + \log y)$ *at the point* $(x,y) = (1,2)$

| $(\cdot)$ | $\left[\frac{\partial}{\partial x}(\cdot), \frac{\partial}{\partial y}(\cdot)\right]$ | returned values |
|---|---|---|
| $t_1 = e^x$ | $[e^x, 0]$ | 2.718 [2.718 0.000] |
| $t_2 = \log y$ | $\left[0, \frac{1}{y}\right]$ | 0.693 [0.000 0.500] |
| $t_3 = \sin(t_1 + t_2)$ | $\left[\cos(t_1+t_2)(\frac{\partial t_1}{\partial x} + \frac{\partial t_2}{\partial x}), \cos(t_1+t_2)(\frac{\partial t_1}{\partial y} + \frac{\partial t_2}{\partial y})\right]$ | -0.267 [-2.620 -0.482] |
| $t_4 = x^2$ | $[2x, 0]$ | 1.000 [2.000 0.000] |
| $z = t_3 t_4$ | $\left[\frac{\partial t_3}{\partial x} t_4 + \frac{\partial t_4}{\partial x} t_3, \frac{\partial t_3}{\partial y} t_4 + \frac{\partial t_4}{\partial y} t_3\right]$ | -0.267 [-3.153 -0.482] |

by systematically applying rules of differentiation to these subproblems, the analytic derivatives of a very complicated function can be automatically computed.

Implementation-wise, Sacado creates new data types that overload elementary operations, returning derivative evaluations as well as performing the requested operation. For example, when requesting the product of the functions of two variables $f(x,y)$ and $g(x,y)$ the multiplication operator * will return the values

$$f * g = \begin{pmatrix} fg \\ f_x g + f g_x \\ f_y g + f g_y \end{pmatrix}, \tag{4.1}$$

where the first entry corresponds to usual scalar multiplication. This structure can also be visualized as the pairing of a scalar value and gradient vector corresponding to the function evaluation and its gradient with respect to the user-specified variables, respectively.

$$f * g = \left\{ fg,\ \left[ f_x g + f g_x,\ f_y g + f g_y \right] \right\} \tag{4.2}$$

Table 4.1 illustrates the evaluation of the function $z(x,y) = x^2 \sin(e^x + \log y)$ at the point $(x,y) = (1,2)$.

Compared to other methods of symbolic and numerical differentiation, Sacado's automatic differentiation technology offers to the developer the convenience of evaluating analytic derivatives of complicated expressions without themselves having to expend a great time in coding and verification. In 3-D MHD modeling on hexahedral grids this is extremely helpful, as each entry in the local mass and stiffness matrix is a nonlinear function of the 24 coordinate values which define the vertices of the element. In this case, we utilize Intrepid's C++ templated data types and declare the matrix entries (see equation 3.3) to be of Sacado's FAD (forward automatic differentiation) type. As will be shown in section 5, however, evaluating these derivatives comes at a high cost.

## 5. Results.

In the following we consider two ideal MHD problems, i.e., ones with no resistive magnetic diffusion step, that also have a negligible thermal pressure (very low beta plasma).

### 5.1. Two-dimensional example.

Given the magnetic potential

$$\mathbf{A}(x,y) = x^2 y^2 \mathbf{k}, \tag{5.1}$$

consider the domain $\Omega = [0,1] \times [0,1]$ tessellated into an initially uniform mesh consisting of $K^2$ squares, where the values $K = 4, 8, 16$ are chosen. In Figure 5.1 the percent energy change (as measured from the initial energy) is plotted as a function of time for the magnetic stress

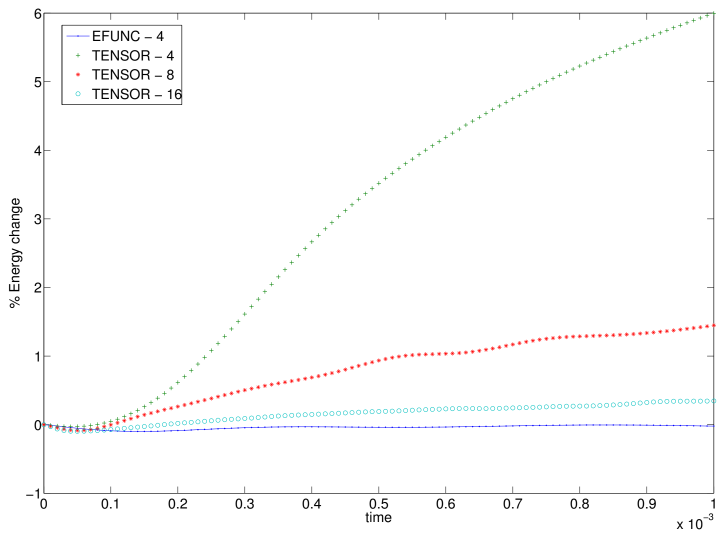


FIG. 5.1. *2D example: change in total energy vs. time. Mesh sizes of* $4^2, 8^2$ *and* $16^2$ *elements are chosen.*

tensor ("TENSOR") and the new energy-based ("EFUNC") force options. The TENSOR option exhibits a growing change in total energy. Under mesh refinement this energy change decreases. The EFUNC option, however, has a relatively constant total energy with less than a 0.1% gain for all time steps.

It is reasonable to expect this small energy fluctuation, since even though the computed Lorentz force is compatible with the discrete energy state of the system at any values of the coordinates, the magnetic energy is a non-linear function of the coordinates. The coordinates $\mathbf{x}$ of an element are updated using a discrete time integration rule using the accelerations given by the energy at a given state. These $\mathbf{B}$-fluxes and scalar potential values for three- and two-dimensional problems, respectively, remain invariant in the Lagrangian motion so at time step $t^n$ we may write

$$\mathbf{x}(\mathbf{\Phi}, t^n) \equiv \mathbf{x}^n.$$

The magnetic energy at the next time step $t^{n+1}$ can be expanded in a Taylor series as

$$E_M(\mathbf{x}^{n+1}) = \sum_{j=0}^{\infty} \left[ \frac{1}{j!} \left( \left( \mathbf{x}^{n+1} - \mathbf{x}^n \right) \cdot \nabla_{\mathbf{x}} \right)^j E_M(\mathbf{x}) \right]_{\mathbf{x}=\mathbf{x}^n} \tag{5.2}$$

The first few terms of this series are

$$E_M(\mathbf{x}^{n+1}) = E_M(\mathbf{x}^n) + \mathbf{h} \cdot \left. \frac{\partial E_M}{\partial \mathbf{x}} \right|_{\mathbf{x}=\mathbf{x}^n} + O(|\mathbf{h}|^2), \tag{5.3}$$

where $\mathbf{h} = \mathbf{x}^{n+1} - \mathbf{x}^n$ measures the mesh displacement. Essentially at each integration step, we are neglecting the higher order terms in the expansion leading to small differences between the magnetic energy and the incremented kinetic energy. These asymptotically reduce to zero under time-step refinement, as is seen in Figure 5.2.

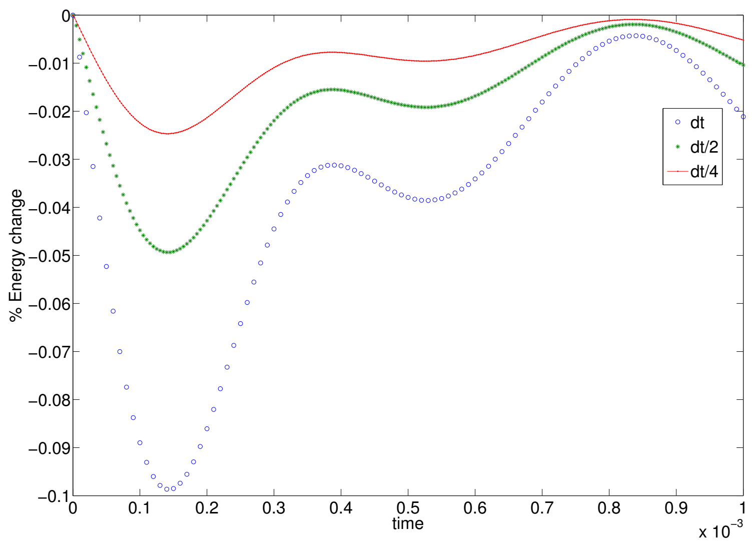


FIG. 5.2. *EFUNC time-step refinement: change in total energy vs time for three different time-step discretizations. A standard time step* dt *is chosen for one case and compared to the same problem run with twice and four times as many time steps. The same 2-D problem as in Figure 5.1 is chosen with a 16-element mesh.*

**5.2. Three-dimensional example.** Given the magnetic potential

$$\mathbf{A}(x, y, z) = \sin^2(xyz)\mathbf{i} + \cos^2(xyz)\mathbf{j} + \tan^2(xyz)\mathbf{k}, \tag{5.4}$$

consider the domain $\Omega = [0, 1] \times [0, 1] \times [0, 1]$ tessellated into an initially uniform mesh consisting of 512 hexahedra. In Figure 5.3 the percent energy change (as measured from the initial energy) is plotted as a function of time for the TENSOR, EFUNC and PJXPBPV options. The latter force option is computed from the projected **J** and **B** quantities. Again as in the two-dimensional case we see the force options increasing in total energy while the new magnetic energy functional option has a relatively negligible change in the total energy.

**5.3. Expense.** Results displaying much improved energy conservation for the new EFUNC force option are promising, but come with the drawback of a much greater computational expense. This can be observed by comparing the CPU time for each option. In Table 5.1 is shown the CPU time spent evaluating the Lorentz force for a two-dimensional problem containing 2000 time steps and a three-dimensional problem containing 350 time steps. All problems assume ideal MHD.

From Table 5.1 we observe a slowdown factor of around 100 in two dimensions and 500 in three dimensions. This cost can be attributed to the fact that with purely Lagrangian motion we must recompute each local mass (or stiffness) matrix and its derivatives for each time step. These numbers make some sense for quadrilaterals because in 2D there are 4 quadrature points in the energy functional, 8 coordinates per element and 3 floating point operations for every multiply in the energy evaluation. This gives a rough cost factor of 96. For hexes a similar computation yields $8 * 24 * 3 = 576$.

At first glance, the cost of the energy-based magnetic force computation is prohibitively high. However, there is room for improvement in the Intrepid and Sacado libraries interface. The symmetry of the mass and stiffness matrices could be utilized and more efficient computations seem to be possible by utilizing specialized code which takes advantage of quad

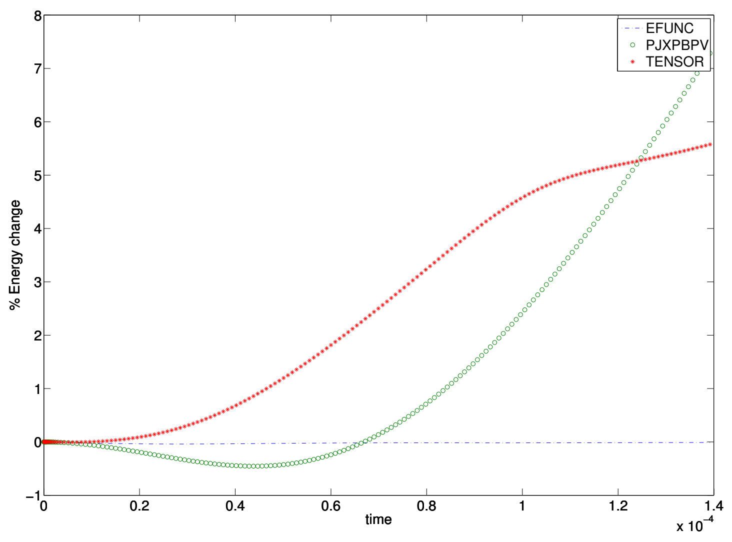


FIG. 5.3. *3D example: change in total energy vs. time. A 512-element mesh is chosen for all three cases.*

TABLE 5.1
*Expense comparison: real-time costs in evaluating the Lorentz force for a two-dimensional problem of 2000 time steps and a three-dimensional problem of 350 time steps. The new EFUNC option is much more costly than the TENSOR option.*

| 2-D problem | | | 3-D problem | | |
|---|---|---|---|---|---|
| Elements | CPU time (s) | | Elements | CPU time (s) | |
| | TENSOR | EFUNC | | TENSOR | EFUNC |
| 16 | 0.12 | 6.69 | 64 | 0.08 | 50.1 |
| 64 | 0.35 | 24.2 | 512 | 0.79 | 401 |
| 256 | 1.09 | 98.3 | 4096 | 6.71 | 3304 |

and hex topologies and specific known relationships for the basis functions and associated derivatives.

In addition to these possible future improvements in performance, the high cost of the new force option is highly apparent in the ideal MHD step. Much of the total cost for real problems of interest actually is related to the magnetic diffusion step which involves solving an implicit system. There is also a significant cost in the ALEGRA remesh/remap phase which is the most common way the code is run. Therefore, for many problems the cost of an ideal MHD step which utilizes the energy based magnetic force methodology may not be as visible and there may be additional advantages for better discrete energy balances which will show up in better results at low resolution.

**6. Summary.** A new option in calculating the Lorentz force in ALEGRA, consistent with the discrete form of the energy used for the magnetic diffusion, has been presented. This method utilizes the Intrepid compatible discretizations library and Trilinos Sacado automatic differentiation technology, two on-going projects at Sandia National Laboratories. Initial results confirm a qualitative improvement in energy conservation over current force options, but the results come at high computational expense. Reducing this cost and understanding the tradeoffs involved remain an open area of research.

**7. Acknowledgments.** We thank Pavel Bochev, Denis Ridzal and David Day for their work in developing Intrepid. Eric Phipps is a lead developer for Sacado and modified Intrepid so that both code bases would work properly together for our use in ALEGRA.

## REFERENCES

[1] P. B. Bochev and A. C. Robinson, *Matching algorithms with physics: Exact sequences of finite element spaces*, in Collected Lectures on the Preservation of Stability under Discretization, SIAM, Philadelphia, 2002, ch. 8.

[2] A. C. Eringen and G. A. Maugin, *Electrodynamics of Continua I and II*, Springer-Verlag, 1990.

[3] M. E. Gurtin, *An Introduction to Continuum Mechanics*, Academic Press, 1981.

[4] T. Mehlhorn, T. Brunner, M. Desjarlais, C. Garasi, T. Haill, H. Hanshaw, R. Lemke, T. Mattsson, M. Matzen, B. Oliver, A. Robinson, S. Slutz, T.G.Trucano, E. Yu, R. Vesey, M. Cuneo, B. Jones, M. D. Knudson, and D. Sinars, *Towards a predictive MHD simulation capability for designing hypervelocity magnetically-driven flyer plates and PW-class z-pinch x-ray sources on Z and ZR*, 2006.

[5] R. Moreau, *Magnetohydrodynamics*, Kluwer Academic Publishers, 1990.

[6] P. W. Rambo, *Resistive MHD for 2D axisymmetric Lagrangian simulation codes*. Unpublished presentation at HEDP-MHD Workshop, Albuquerque, New Mexico, January 14, 1998.

[7] A. C. Robinson and C. J. Garasi, *Three-dimensional z-pinch wire array modeling with ALEGRA-HEDP*, Computer Physics Communications, 164 (2004), pp. 408–413.

# EFFECT OF ASPHERICITY ON THE DIFFUSION OF SOLUTES IN A LENNARD-JONES SOLVENT

BABATUNDE O. OGUNTADE[*] AND STEVEN J. PLIMPTON[†]

**Abstract.** This work focused on studying the effects of shape on the behavior of colloidal suspensions. Specifically, the rotational and translational diffusivities of aspherical particles in solvents were measured, with the solvent approximated as a Lennard-Jones fluid. Recent additions to the LAMMPS molecular dynamics package were used which enable the pairwise interactions of two ellipsoidal or an ellipsoidal and spherical particle to be computed via the Gay-Berne potential. The preliminary results indicate that shape has a modest effect on diffusivity, though we limited our study to ellipsoids with an aspect ratio no larger than 3. This was because the distance-of-closest-approach computation needed for Gay-Berne interactions of ellipsoids and spheres (solvent) becomes less accurate beyond that limit. Future work on a more robust algorithm to perform this computation could extend these studies to higher aspect ratio ellipsoids.

## 1. Introduction.

Colloids are dispersions of particles in a continuous medium. They are ubiquitous in nature and also find application in industrial settings. Typical colloidal particles in suspension span a length scale of 1 nanometer to 1 micron and are found in materials such as aerosols, emulsions, foams, etc. An assumption made in most computational models of such systems is that the colloidal particles are spherical. This simplifies the analysis but could obscure mechanical, structural, and dynamic properties specific to suspensions of aspherical particles.

As a first step in understanding the effect of shape on the dynamic behavior of suspensions, a pair potential designed for aspherical particles can be used, such as the Gay-Berne potential, originally devised for liquid-crystal polymer (LCP) systems. This option is attractive, because LCPs are of comparable size to the smallest of colloidal particles and LCP mixtures exhibit properties like birefringence and liquid phase changes which have counterparts in colloidal systems as well. From a computational standpoint, the Gay-Berne potential can also be computed for an ellipsoid interacting with a spherical particle, which means a solution of colloidal particles in a traditional solvent can be simulated directly.

We note that the comparison between LCP and colloidal systems breaks down as the size of solute particles increase because of increased time of interaction which could result in increased relaxation times. Hydrodynamic effects of the background solution on large colloidal particles is also expected to be more pronounced. Notwithstanding these drawbacks, this project modeled the diffusive behavior of small colloidal particles (1 nm) dispersed in an LJ solvent using the Gay-Berne potential.

The objectives were to (1) measure the transport coefficients of colloidal suspensions using LCP-based potentials; (2) determine the applicability of LCP-based potentials to colloidal suspensions by comparing the numerical solutions to known experimental and numerical values; and (3) assess the applicability of an integrated form of LCP based potentials for suspensions of larger colloidal particles.

The simulations discussed in this paper were run with the LAMMPS parallel molecular dynamics package [4].

## 2. Measured Properties.

The transport properties measured in this study are the rotational and translational diffusivities of prolate and oblate spheroids as in Figure 2.1. The rotational diffusion coefficient is a measure of how quickly particles relax to a state uncorrelated with their initial orientation, while the translational diffusion coefficient is a measure of the mobility of particles moving in a translational sense through the background fluid.

[*]University of Texas at Austin, tade@utexas.edu
[†]Sandia National Laboratories, sjplimp@sandia.gov

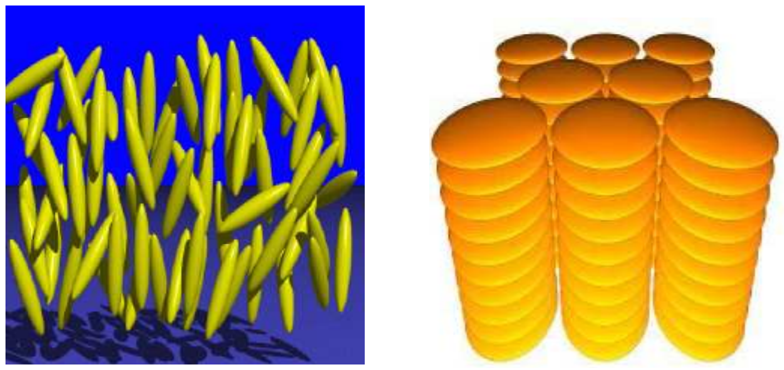


FIG. 2.1. *Prolate (left) and Oblate (right) Spheroids*

The rotational diffusivity is measured by tracking changes in the orientation of individual ellipsoids. The orientation is the direction of the longest axis for prolate spheroids; for oblate spheroids it is the direction of the shortest axis. The Green-Kubo relation for the dependence of the rotational diffusivity coefficient on orientation is given by

$$< p(t+\tau) \bullet p(t) >= \exp(-2D_R) \tag{2.1}$$

where $p$ is the orientation vector of a particle, $D_R$ is the rotational diffusivity and $<>$ denotes an average. The dot product of time-varying $p$ with an initial reference $p$ exhibits a decay from 1.0 (correlated) to 0.0 (uncorrelated), the time constant of which is inversely proportional to $D_R$.

The translational diffusivity is measured by tracking the mean-squared displacement (MSD) of individual particles over time. For a three-dimensional system, the Green-Kubo relation connecting MSD to a translational diffusion coefficient is

$$< (x(t+\tau) - x(t)) \bullet (x(t+\tau) - x(t)) >= 6D_T\tau \tag{2.2}$$

where $x$ is the position vector and $D_T$ is the diffusion coefficient. For a system of freely diffusing particles, the MSD increases linearly in time and the slope of the change is proportional to $D_T$.

LAMMPS already had the capability of measuring $D_T$ during a simulation. We coded a new diagnostic capability for $D_R$ measurement as part of this project and added it to LAMMPS. Both the diagnostics perform appropriate averaging over particles and time to compute statistically averaged values of these diffusion coefficients for a particular simulation run.

## 3. Gay-Berne Potential.

The Gay-Berne potential, a recent addition to LAMMPS by Mike Brown at Sandia, computes an anisotropic LJ interaction [2] between pairs of ellipsoidal particles or an ellipsoidal and spherical particle via the formulas

$$U(\mathbf{A}_1, \mathbf{A}_2, \mathbf{r}_{12}) = U_r(\mathbf{A}_1, \mathbf{A}_2, \mathbf{r}_{12}, \gamma) \cdot \eta_{12}(\mathbf{A}_1, \mathbf{A}_2, \upsilon) \cdot \chi_{12}(\mathbf{A}_1, \mathbf{A}_2, \mathbf{r}_{12}, \mu)$$

$$U_r = 4\epsilon(\varrho^{12} - \varrho^{6})$$

$$\varrho = \frac{\sigma}{h_{12} + \gamma\sigma}$$

where $A_1$ and $A_2$ are the transformation matrices from the simulation box frame to the body frame and $r_{12}$ is the center-to-center vector between the particles. $U_r$ controls the shifted distance-dependent interaction based on the distance-of-closest-approach of the two particles ($h_{12}$) and the user-specified shift parameter $\gamma$. When both particles are spherical, the formula reduces to the usual Lennard-Jones interaction.

The first term in the above formula accounts for the dependence of the interaction on distance where the interparticle distance is replaced by the distance-of-closest-approach. The second term captures the interaction dependence on the orientation of the particles, while the third term defines the dependence of the interaction on position. The interaction strength between the particles is usually stronger in the side-to-side configuration. A fuller discussion of the $\upsilon$ and *mu* variables in this formula, and the details of the $\eta_{12}$ and $\chi_{12}$ terms, is outside the scope of this short paper. The interested reader is referred to the [2] and [3] references and to the explanatory document `http://lammps.sandia.gov/doc/Eqs/pair\_gayberne\_extra.pdf` written by Mike Brown, which is part of the on-line LAMMPS documentation.

For large uniform molecules it has been shown that the usual LJ energy parameters $\epsilon$ for interactions between ellipsoidal particles aligned on different axes $a, b, c$ are approximately representable in terms of local contact curvatures as

$$\epsilon_a = \sigma \cdot \frac{a}{b \cdot c}; \epsilon_b = \sigma \cdot \frac{b}{a \cdot c}; \epsilon_c = \sigma \cdot \frac{c}{a \cdot b}$$

This formulation is also consistent with the Everaer paper's RE-squared potential [3]. We mention this because the Everaers approach for creating interaction potentials between pairs of large particles by integrating over a collection of small Lennard-Jones particles is compatible with the Gay-Berne formulation and is a mechanism for deriving potentials for larger colloidal particles, including aspherical particles. This extended approach is one we are planning to implement in LAMMPS as an extension to the work discussed here.

When aspherical particles interact via the Gay-Berne potential, both forces and torques on the particles are produced, via the spatial derivatives of these formulas. Special integration options are available in LAMMPS to evolve the translational and angular velocity of each particles as well as their position and orientation. The latter is represented for aspherical particles via quaternions, which provide numerical stability when time integration is performed [1].

**4. Simulation Details.** The simulation boxes for simulations of both prolate and oblate particles are illustrated in Figure 4.1. The domain is periodic in all three dimensions. The aspect ratio of the ellipsoidal particles was set at 3.

The asphericity of the particles dictates that the pairwise interactions have some form of anisotropy. We specify this in our model by setting the $\epsilon$ for interactions in the side-to-side configuration to be 5x stronger than for end-to-end interactions, which is a typical of other published models using Gay-Berne potentials for particles with aspect ratios around 3.

The relative sizes of the particles were chosen so that the volumes of prolate and oblate spheroids were equal. For a desired volume fraction of ellipsoidal particles, the system was initialized in the following manner. A lattice of spherical particles was created at a very low density. Some fraction of these were converted randomly to ellipsoidal particles, each at a random orientation. The low density insured that even with elongated particles, no overlaps

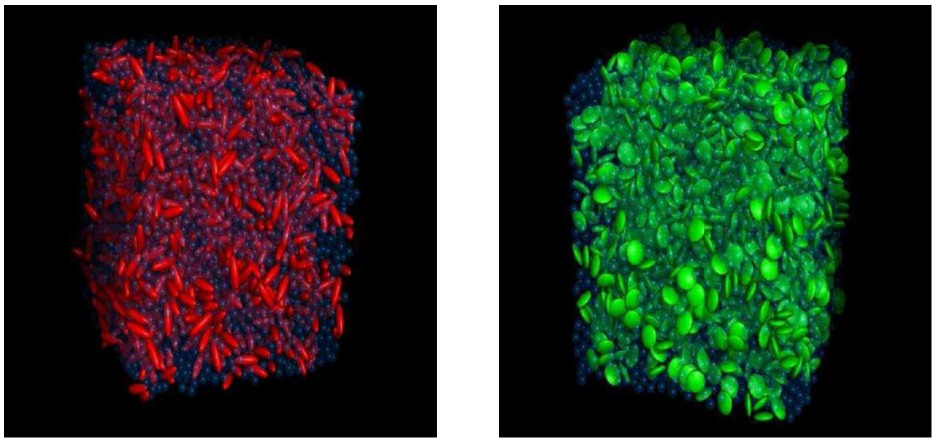


FIG. 4.1. *Simulation box showing prolate (red) and oblate (green) spheroids in a Lennard-Jones solvent (blue).*

occurred. This system was run in a constant NPT ensemble to shrink the volume of the box to a desired solvent reduced density, using a reduced LJ timestep of 0.00025. The dense system was then run for an additional 10,000 timesteps in the NVT ensemble to equilibrate it. The thermostat was then turned off and a constant NVE run of 150,000 steps was performed to generated the translational and rotational data for the two diffusion coefficient. For rotational diffusivity, the results for the first 20000 timesteps (after equilibration) were used in calculating the coefficient as it is a short-time phenomenon. Data for the entire run was used in estimating the translational diffusivity.

A collection of runs with volume fractions varying from 0.002 to 0.18 were performed for both shapes. The total number of particles in a simulation (solute and solvent) was approximately 10648. The simulation runs were performed on the CSRI's "qed" machine, typically running on 16 processors. A typical simulation for a single state point ran in an hour or two of wall-clock time, so larger and longer simulations are planned.

**5. Results.** The plots of Figures 5.1 and 5.2 show results for the two diffusion coefficients as a function of volume fraction. Each data point is from a 150,000 timestep run as described in the previous section.

Translational diffusivity decreases linearly with the volume fraction of both prolate and oblate spheroids. The values of the coefficients computed for both spheroids are similar for the concentration regime investigated. The import of this is that aspherical particles are less mobile at high volume fractions, but individual particle shape has less effect on the translational diffusivity. How general this conclusion is remains to be seen when higher aspect ratio ellipsoids are simulated. The rotational diffusivity data exhibit a weaker dependence on volume fraction, but the oblate particles have a consistently higher rotational diffusivity than their prolate counterparts, meaning they can spin or rotate more rapidly in the dense surrounding fluid.

**6. Conclusions.** The effect of shape on the rheology of suspensions is a largely unexplored area of colloidal research. We have measured the translational and rotational diffusivity of spheroids (volume fraction 0.02-0.18) in a model solvent across a range of reduced densities of 0.66-0.73. The aspect ratio of the spheroids were held at 3 because of the current restriction on our optimization routine that evaluates the distance-of-closest-approach used in the Gay-Berne potential in calculating the interaction potential. We intend to make the

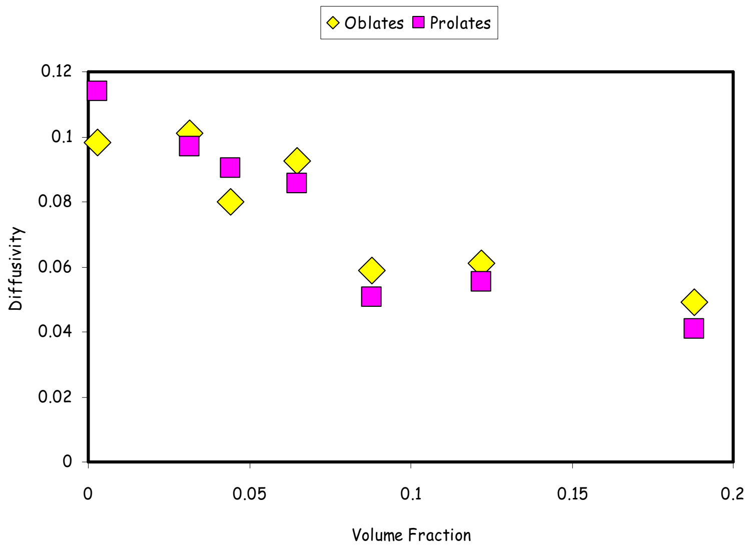


FIG. 5.1. *Translational diffusivity coefficient dependence on volume fraction of aspherical particles*

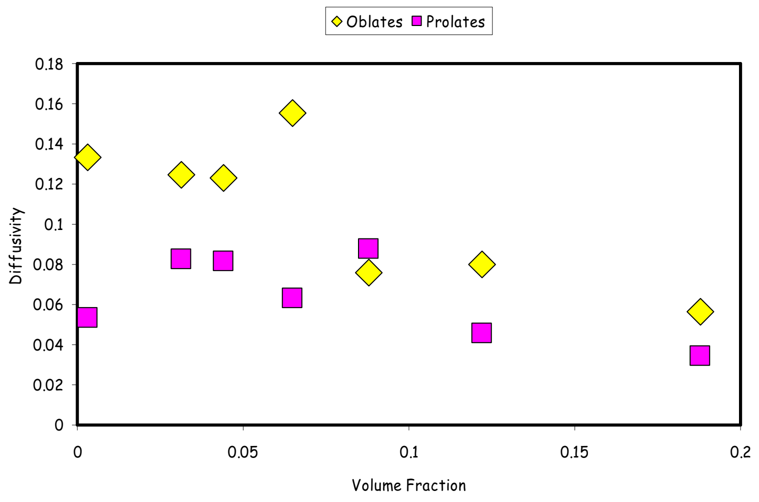


FIG. 5.2. *Rotational diffusivity coefficient dependence on volume fraction of aspherical particles*

routine more robust for studies involving higher aspect ratio ellipsoids.

In summary, both the translational and rotational diffusivity coefficients decreased linearly with volume fraction, though the dependence was stronger for the translational coefficients. Translational diffusivity coefficients were found to be independent of the shape of the spheroids within the range of concentration probed. For the rotational case, oblate spheroids had larger diffusivities than prolate spheroids by up to a factor of 2x. The results obtained thus far look qualitatively reasonable, making the application of Gay-Berne potential models to aspherical colloidal suspensions promising. Longer runs need to be performed to quantify the error bars on the diffusivity coefficients, and the computed values still need to be compared to theoretical and experimental measurements to validate our overall approach. Once this is done, an integrated version of the Gay-Berne potential could be used for large interacting aspherical particles to address a class of problems hitherto intractable. Additionally, for realistic simulations of colloidal suspensions containing large particles at the micron size scale, a "coarse grained" approach must be used to include solvation effects, since the number of small solvent particles will otherwise be computationally prohibitive. For this reason, we are also working on adding new coarse-grained solvent models to LAMMPS.

**7. Acknowledgements.** We thank Randy Schunk and Jeremy Lechman at Sandia for additional support and fruitful discussions on these topics.

## REFERENCES

[1] M. P. Allen and D. J. Tildesley, *Computer Simulation of Liquids*, Clarendon Press, Oxford, 1987.

[2] C. R.Berardi and C.Zannoni, *A Gay-Berne potential for dissimilar biaxial particles*, Chem Phys Lett, 297 (1998), pp. 8–14.

[3] R.Everaers and M. Ejtehadi, *Interaction potentials for soft and hard ellipsoids*, Phys. Rev. E, 67 (2003), pp. 041710–041710–8.

[4] S.Plimpton, *Fast parallel algorithms for short-range molecular dynamics*, J. Comp. Phys, 117 (1995), pp. 1–19. See also the LAMMPS WWW site at `http://lammps.sandia.gov`.

# A STUDY OF SOLVERS FOR FLUID DENSITY FUNCTIONAL THEORIES

SARAH M. KNEPPER* AND MICHAEL A. HEROUX†

**Abstract.** The use of density functional theories for inhomogeneous fluids (Fluid-DFTs) creates large systems of residual equations by minimizing a free energy functional that is dependent on a set of density fields. Each system of equations can be viewed as one large, global system and solved in that manner, either by an iterative or a direct method. Alternatively, for certain types of problems, by intelligently ordering the degrees of freedom, a two-by-two block structure may be imposed and a Schur complement computed, reducing the run-time and memory requirements. Previously, the Tramonto package [3] allowed CMS-polymers [1] without Coulomb effects to be solved in this more efficient manner (see [7]). We will now consider how adding Coulomb effects changes the structure of the sub-matrices, though still allowing the use of a Schur complement approach.

**1. Introduction.** Density functional theories (DFTs) are useful for a variety of situations at different length scales, including predicting electron distribution, studying self-assembly, and observing biological mechanisms at the cellular level. Segregated solvers, which separate each degree of freedom (DOF), can be used in conjunction with a Schur complement approach, which eliminates variables by using block Gaussian elimination. This can allow DFT problems to be solved more efficiently than using a global approach, especially in the case of polymers, on which we focus. In particular, we will consider how the introduction of Coulomb effects on CMS-polymers affects the block structure of the global matrix.

We briefly introduce Fluid-DFTs in Section 2, with Section 3 describing the segregated Schur complement strategy for CMS-polymers. The block structure with the addition of Coulomb effects is presented in Section 4, and results are given in Section 5. Finally, Section 6 discusses future work in this area while Section 7 summarizes the conclusions of the research presented here.

**2. Brief Overview of Fluid-Density Functional Theory.** The theory behind Fluid-DFTs is exact, but approximations must be used. Specifically, approximate functionals have been developed as perturbations to a hard sphere reference system. The free energy functional $\Omega$, which depends on a set of density fields $\rho_i(\mathbf{r})$, can be split into ideal ($F_{id}$), hard sphere ($F_{hs}$), and perturbation ($F_p$) contributions. For a system with $N_i$ species, we have

$$\Omega = F_{id} + F_{hs} + F_p - \sum_{i=1}^{N_i} \int \rho_i(\mathbf{r}) \left[ V_i^{ext}(\mathbf{r}) - \mu_i \right] d\mathbf{r}, \tag{2.1}$$

where $\mathbf{r}$ are the fluid particles, $V_i^{ext}$ is the known external field acting on species $i$, and $\mu_i$ is the chemical potential of species $i$. Please see [4, 5, 6, 7, 10] for more information about these and following terms. We wish to find the global free energy minimum:

$$\frac{\delta\Omega}{\delta\rho_i(\mathbf{r})} = 0. \tag{2.2}$$

For our polymer problems, we use the CMS-DFT crafted by Chandler, McCoy, and Singer [1]. This theory is developed by minimizing a free energy functional with respect to both the density fields and an unknown field $U$. The unknown field variable requires a fluid of ideal chains to have exactly the same density profile as do the chains of interest in a known external field $V^{ext}$. The density equation is

*Emory University, smknepp@emory.edu
†Sandia National Laboratories, maherou@sandia.gov

$$\rho_i = \frac{\rho_{b,i}}{N_i} \sum_{s=1}^{N_s} \frac{G_s(\mathbf{r}) G_s^{inv}(\mathbf{r})}{e^{-\beta U_{\beta(s)}(\mathbf{r})}}, \tag{2.3}$$

where the sum over $s$ is a sum over all $N_s$ segments of type $i$ in a chain, $\beta(s)$ represents the bead type associated with segment $s$, and $G$ and $G^{inv}$ are propagator functions that describe chain connectivity.

The unknown field equations are

$$U_i(\mathbf{r}) = V_i^{ext}(\mathbf{r}) - \sum_{\beta} \int c_{i\beta}(\mathbf{r} - \mathbf{r}')(\rho_\beta(\mathbf{r}') - \rho_{b\beta}) d\mathbf{r}', \tag{2.4}$$

where $c_{i\beta}$ is the direct correlation function between sites $i$ and $\beta$ in the bulk fluid and $\rho_{b\beta}$ is the bulk density for site type $\beta$. The propagator functions obey the recursion relations

$$G_s(\mathbf{r}) = e^{-\beta U_{i(s)}(\mathbf{r})} \int \omega_{12}(\mathbf{r} - \mathbf{r}') G_{s-1}(\mathbf{r}') d\mathbf{r}', \quad s = 2 \ldots N_s, \tag{2.5}$$

$$G_s^{inv}(\mathbf{r}) = e^{-\beta U_{i(s)}(\mathbf{r})} \int \omega_{12}(\mathbf{r} - \mathbf{r}') G_{s+1}^{inv}(\mathbf{r}') d\mathbf{r}', \quad s = (N_s - 1) \ldots 1, \tag{2.6}$$

where $\omega_{12}$ is a delta function. We note that $G_1 = e^{-\beta U_{i(1)}(\mathbf{r})}$ and $G_{N_s}^{inv} = e^{-\beta U_{i(N_s)}(\mathbf{r})}$.

**3. Block Equation Framework with Respect to Polymers.** As the equations show, there are many DOFs per node, though internodal coupling is weak. Many of the DOFs exhibit a one-way dependence. The main idea is to organize the DOFs physics-first in such a way that a two-by-two block structure is imposed on the matrix as

$$\left(\begin{array}{ccc|ccc}
A_{11}^{1,1} & \cdots & A_{11}^{1,j} & A_{12}^{1,j+1} & \cdots & A_{12}^{1,k} \\
\vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\
A_{11}^{j,1} & \cdots & A_{11}^{j,j} & A_{12}^{j,j+1} & \cdots & A_{12}^{j,k} \\
\hline
A_{21}^{j+1,1} & \cdots & A_{21}^{j+1,j} & A_{22}^{j+1,j+1} & \cdots & A_{22}^{j+1,k} \\
\vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\
A_{21}^{k,1} & \cdots & A_{21}^{k,j} & A_{22}^{k,j+1} & \cdots & A_{22}^{k,k}
\end{array}\right)
\left(\begin{array}{c} x_1^1 \\ \vdots \\ x_1^j \\ \hline x_2^{j+1} \\ \vdots \\ x_2^k \end{array}\right)
=
\left(\begin{array}{c} b_1^1 \\ \vdots \\ b_1^j \\ \hline b_2^{j+1} \\ \vdots \\ b_2^k \end{array}\right) \tag{3.1}$$

where $k$ is the number of DOFs per node and $j$ is the number of DOFs associated with the first block of equations. The coefficients generated by DOF $p$ interacting with DOF $q$ are found in the appropriate $A^{p,q}$ subblock. We will refer to the northwest portion of the matrix as $A_{11}$, the northeast as $A_{12}$, the southwest as $A_{21}$, and the southeast as $A_{22}$. Similarly, $x_1$ and $b_1$ are the upper, and $x_2$, $b_2$ the lower, parts of $x$ and $b$, respectively.

The basic solving strategy for each linear system generated by Newton's method then becomes:

1. Reorder the DOFs so that $A_{11}$ has all the one-way dependencies, making its inverse easy to apply in parallel. Specifically, Eqs. (2.5) and (2.6), the $G$ and $G^{inv}$ equations, belong here. The density, Eq. (2.3), and the unknown field, Eq. (2.4), equations belong in $A_{22}$.

2. Use an approximation of $A_{22}$ for a preconditioner $P$ for $S = A_{22} - A_{21}A_{11}^{-1}A_{12}$, the Schur complement of $A$ with respect to $A_{22}$. See Section 4 for more details.
3. Solve $Sx_2 = (b_2 - A_{21}A_{11}^{-1}b_1)$ using an iterative solver like GMRES with preconditioner $P$.
4. Solve $A_{11}x_1 = (b_1 - A_{12}x_2)$, realizing that $A_{11}$ was constructed so its inverse is easy to apply.

**4. Addition of Coulomb Effects.** To introduce Coulomb effects into the polymer model, we must add the following to the left-hand side of Eq. (2.2):

$$z_i e\left(V^C + \sum_j \int \frac{\rho_j(\mathbf{r}')z_j e}{|\mathbf{r}-\mathbf{r}'|}d\mathbf{r}'\right) = z_i e\phi(\mathbf{r}) \tag{4.1}$$

where $z_i$ is the charge of species $i$ and $V^C$ is the Coulombic part of the known external field ($V^{ext}$). We must also solve Poisson's equation:

$$\nabla^2\phi(\mathbf{r}) + \sum_i z_i e\rho_i(\mathbf{r}) = 0. \tag{4.2}$$

Eq. (2.4) must also be amended to include Coulomb interactions. Since the unknown field equations interact with the Coulomb equations, and the Coulomb equations interact with the primitive density equations, the inclusion of Coulomb effects introduces three new matrices. An immediate question is where should the Coulomb-on-Coulomb matrix (hereafter referred to as the Coulomb matrix) be placed? There are two possible answers:

1. In $A_{11}$, where it can be solved directly. The $G$ and $G^{inv}$ equations and the Coulomb equations do not interact with each other. The other two matrices would become part of the off-diagonal $A_{12}$ and $A_{21}$ blocks. This location may be more appropriate if the problem is 1- or 2-dimensional and run on few processors. However, the Coulomb matrix may be very ill-conditioned or even singular, so a direct solve may not be possible.
2. Or in $A_{22}$, where multi-level preconditioning can be applied. The other two matrices would also go into the $A_{22}$ block. This may be the more appropriate choice for a 3-dimensional problem or one with many processors.

Since the addition of the Coulomb matrix to the $A_{11}$ block is fairly straightforward, we will not go into any implementation details. However, we will look more closely at including the Coulomb matrix in the $A_{22}$ block. Specifically, our $A_{22}$ block will look like the following:

$$A_{22} = \begin{pmatrix} B_{11} & B_{12} & 0 \\ 0 & B_{22} & B_{23} \\ B_{31} & B_{32} & B_{33} \end{pmatrix} \tag{4.3}$$

where:

- $B_{11}$ is the Coulomb matrix, a Poisson-like matrix of dimension $n$, where $n$ is the number of nodes in the system,
- $B_{12}$ is the Coulomb-on-density matrix, consisting of $N_i$ finite-element mass matrices, each scaled by the charge of its corresponding component ($z_i$),
- $B_{22}$ is the density-on-density matrix, a diagonal matrix of dimension $N_i \times n$,
- $B_{23}$ is the density-on-unknown field matrix, a diagonal matrix of the same size as $B_{22}$, but whose elements are orders of magnitude smaller than $B_{22}$ or $B_{33}$,

- $B_{31}$ is the unknown field-on-Coulomb matrix, consisting of $N_i$ diagonal matrices, each with the corresponding $z_i$ on the diagonal,
- $B_{32}$ is the unknown field-on-density matrix, also known as the $F$ matrix in [7]; this is the most dense submatrix in the entire matrix, and
- $B_{33}$ is the unknown field-on-unknown field matrix, a diagonal matrix of the same size as $B_{22}$.

To help visualize these different submatrices, Figure 4.1 shows the $A_{22}$ block of a 1-dimensional, 3-component polymer system with Coulomb effects; the third component is a neutral solvent (charge of 0).

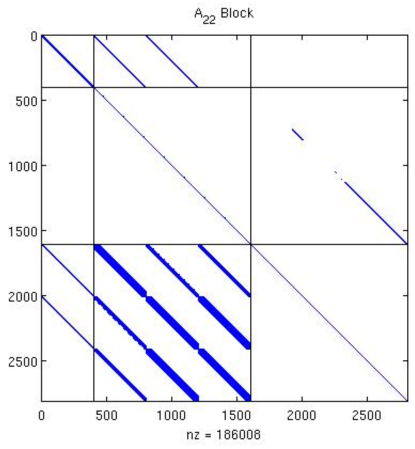


FIG. 4.1. *Visualization of $A_{22}$ for a 3-component system*

As stated in Section 3, we use an approximation of the $A_{22}$ block to precondition $S$. Since the values of $B_{23}$ are relatively very small, we approximate $A_{22}$ by setting $B_{23}$ to zero and leaving the other submatrices unchanged. We then perform one block Gauss-Seidel back-solve. That is,

$$\begin{pmatrix} B_{11} & B_{12} & 0 \\ 0 & B_{22} & 0 \\ B_{31} & B_{32} & B_{33} \end{pmatrix}^{-1} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} \tag{4.4}$$

where

- $y_2 = B_{22}^{-1} x_2$, a diagonal scaling,
- $y_1 = B_{11}^{-1}(x_1 - B_{12} y_2)$, which requires the use of a multi-level preconditioner from the ML package of Trilinos [8], and
- $y_3 = B_{33}^{-1}(x_3 - B_{31} y_1 - B_{32} y_2)$, two matrix-vector multiplications followed by a diagonal scaling.

Using the Schur complement approach is much faster and more memory-efficient than solving a global matrix, as will be shown in Section 5.1.

## 5. Computational Results.

We will now present results for two different types of tests. First, we will demonstrate the effectiveness of the Schur complement method in comparison to a global solving approach. However, since we only have one test problem, our results will be a little meager. We will also offer a comparison of solving times for an iterative solver versus a direct solver for the global matrix on a wide range of test problems. All of the tests were run on an Intel Core 2 computer with a 2.4 GHZ clock rate.

**5.1. Schur Approach versus Global Approach.** This problem considers a CMS-polymer system with Coulomb effects. Three components make up this 8-2-8 lipid bilayer; the tail segments carry a negative charge while the two head beads carry a positive charge. The third component, the solvent, is neutral. We will consider two different sets of boundary problems for this 1-dimensional problem:

1. both of the boundary conditions are reflective, and
2. the left boundary condition is a continuation boundary while the right boundary condition is reflective.

Table 5.1 compares the Schur complement approach ($S$) to the global approach ($G$). A level-4, zero drop-tolerance ILUT preconditioner with row-sum scaling was used for the global matrix. As is apparent from the table, the Schur approach is considerably faster, especially as the number of processors increases. This is due to the fact that the problem is difficult to solve, especially when preconditioning must be applied in parallel for the global solve.

Performance results for the Schur approach alone are given in Table 5.2. This table compares the number of Newton nonlinear iterations required for different numbers of processors. As can be seen, the number of nonlinear iterations generally increases slightly with the processor count, though the average number of linear iterations per nonlinear iteration decreases slightly.

These results were given with the Coulomb matrix in the $A_{22}$ part of the matrix, as described in Section 4. We also tried including the Coulomb matrix in the $A_{11}$ part of the matrix; however, due to its singularity, the direct solver would fail. A potential solution to this was to add $10^{-12}$ to the diagonal, effectively changing the steady-state problem into a transient problem with a time step of $10^{12}$. Though this did allow convergence with certain parameters, it would fail on multiple processors. Thus, we have no further results for placing the Coulomb matrix into the $A_{11}$ block.

TABLE 5.1

*Comparison between the Schur complement approach, denoted by $S$, and the general, global approach, denoted by $G$, for the polymer problem described above. The columns are: the number of processors used, the number of nonlinear iterations, the average number of linear iterations per nonlinear iteration, the solve time (in seconds) per nonlinear iteration, and the ratio of solve times from the global method to the Schur method. The data in the upper part correspond to the first set of boundary conditions while the data in the lower part correspond to the second set.*

| # Procs | $N_{iter}$ | | $L_{iter}$ | | $T/N_{iter}$ | | $T_G$ / $T_S$ |
|---|---|---|---|---|---|---|---|
| | $S$ | $G$ | $S$ | $G$ | $S$ | $G$ | |
| 1 | 22 | 5 | 78 | 79 | 0.87 | 6.27 | 1.63 |
| 2 | 23 | 17 | 51 | 84 | 0.51 | 3.61 | 5.25 |
| 1 | 22 | 7 | 49 | 82 | 0.69 | 6.39 | 2.95 |
| 2 | 25 | 23 | 53 | 58 | 0.52 | 2.68 | 4.72 |

**5.2. Iterative versus Direct Solvers for Global Matrix.** We now consider the general method for solving. We compare the time required to solve the global matrix via an iterative method, namely GMRES with a level-4 ILUT preconditioner with zero drop-tolerance, to the time taken for a direct method, using an Amesos solver with SuperLU [2]. A total of 42 tests were run; on one processor, the direct solver was faster in 26 of the tests, or 62% of the time. When two processors were used, however, the direct solver was faster only about 12% of the time, in 5 tests. The direct solver was not able to solve the largest problem (incidentally, that is the problem described in the previous section). Figure 5.1 shows a plot of the timings on one processor. The tests are sorted by the total number of unknowns (the number of

TABLE 5.2
*A comparison of iterations for different number of processors for the polymer problem described above. The columns are: the number of processors, the number of nonlinear iterations, and the average number of linear iterations per nonlinear iteration. The left part of the table corresponds to the first set of boundary conditions while the right part corresponds to the second set. The asterisk (*) denotes a problem that converged after the given number of iterations but did not complete its run.*

| # Procs | $N_{iter}$ | $L_{iter}$ | $N_{iter}$ | $L_{iter}$ |
|---|---|---|---|---|
| 1 | 22 | 78 | 22 | 49 |
| 2 | 23 | 51 | 25 | 53 |
| 3 | 24 | 45 | 24 | 42 |
| 4 | 27 | 36 | 23 | 37 |
| 5 | 26 | 36 | 23* | 36 |
| 6 | 200+ | | 200+ | |

unknowns per node multiplied by the number of nodes). However, the number of unknowns is not necessarily an indicator of the "toughness" of a problem. In general, the direct solver did well for non-polymer problems.

Table 5.3 shows the times for seven specific tests. The 42 tests were separated into seven groups, and the results from an average test from each group is presented. As can be seen, the savings of using a direct solver are sometimes quite impressive, though again, we must remember that this is with only one processor.

One may also wonder about the effectiveness of using a distributed direct solver. Timings were also obtained for Amesos with SuperLU_DIST [9]; however, most of the test problems were not large enough to offset the costs of using SuperLU_DIST. On a side note, the problem not solvable by SuperLU was able to be solved, and quicker than the iterative solver, with SuperLU_DIST. However, more tests on larger problems would need to be run to determine the effectiveness of using a direct solver in parallel.

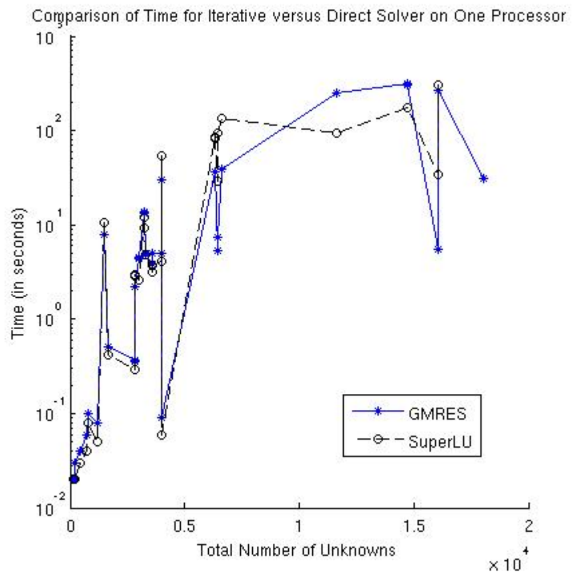


FIG. 5.1. *Timings comparison for iterative (GMRES) and direct (SuperLU) solvers for 42 test problems, ranked by number of unknowns*

TABLE 5.3

*Comparison of time, in seconds, required for seven example problems to be solved iteratively or directly. Hard sphere is abbreviated HS while LJ stands for Lennard-Jones.*

| Problem Type | Iterative Time | Direct Time |
|---|---|---|
| 2-D Charged HS | 4.90 | 4.86 |
| 3-D Charged HS | 307.20 | 172.84 |
| 1-D HS near wall | 0.06 | 0.04 |
| 1-D LJ Fluid | 2.19 | 2.91 |
| 1-D Electrolyte | 5.03 | 4.09 |
| 1-D 3-Polymer | 7.99 | 10.65 |
| 1-D 5-Polymer | 36.85 | 84.13 |

**6. Future Work.** A number of opportunities exist for future work to be done. First, we would like to determine a way to include the Coulomb matrix in the $A_{11}$ block that allows for a direct solve on multiple processors, or determine a way to verify that such an approach is impossible for a given problem. We would also like to run more test problems, especially those having both CMS-polymers and Coulomb effects, on varying numbers of processors. Finally, we would like to take advantage of the parts of the matrix that do not change between nonlinear iterations; namely, these are blocks $B_{11}$, $B_{12}$, $B_{31}$, and $B_{32}$ from Eq. (4.3).

**7. Conclusions.** In this paper we presented a segregated Schur complement approach to solving CMS-polymer problems, particularly those with Coulomb effects. We observed that solving with this method instead of a global approach is typically faster and, though we did not provide results showing it, also more memory efficient. Additionally, we looked at solving the global matrix with a direct solver instead of an iterative one and found that there may be savings, especially when using one processor. We look forward to applying our research to further test cases.

**8. Acknowledgements.** We would like to acknowledge the continual assistance of Andy Salinger for his help with Tramonto and DFTs throughout this project. Additionally, the suggestions given by Chris Siefert for ML parameters have proved highly valuable. Finally, the first author would like to acknowledge the constant support and help of Mike Heroux, who is truly a wonderful mentor.

## REFERENCES

[1] D. Chandler, J. D. McCoy, and S. J. Singer, *Density functional theory of nonuniform polyatomic systems. I. General formulation*, J. Chem. Phys., 85 (1986), pp. 5971–5976.

[2] J. W. Demmel, S. C. Eisenstat, J. R. Gilbert, X. S. Li, and J. W. H. Liu, *A supernodal approach to sparse partial pivoting*, SIAM J. Matrix Analysis and Applications, 20 (1999), pp. 720–755.

[3] L. Frink, *Tramonto home page*, `http://software.sandia.gov/tramonto`, (2007).

[4] L. J. D. Frink and A. G. Salinger, *Two- and three-dimensional nonlocal density functional theory for inhomogeneous fluids I. Algorithms and parallelization*, J. Chem. Phys., 159 (2000), pp. 407–424.

[5] ———, *Rapid analysis of phase behavior with density functional theory. II. Capillary condensation in disordered porous media*, J. Chem. Phys., 118 (2003), pp. 7466–7476.

[6] L. J. D. Frink, A. G. Salinger, M. P. Sears, J. D. Weinhold, and A. L. Frischknecht, *Numerical challenges in the application of density functional theory to biology and nanotechnology*, J. Phys.-Cond. Matter, 14 (2002), pp. 12167–12187.

[7] M. A. Heroux, A. G. Salinger, and L. J. D. Frink, *Parallel segregated Schur complement methods for fluid density functional theories*, SIAM SISC, 29 (2007), pp. 2059–2077.

[8] M. A. Heroux and J. M. Willenbring, *Trilinos Users Guide*, Tech. Report SAND2003-2952, Sandia National Laboratories, 2003.

[9] X. S. Li and J. W. Demmel, *SuperLU_DIST: A scalable distributed-memory sparse direct solver for unsymmetric linear systems*, ACM Trans. Mathematical Software, 29 (2003), pp. 110–140.

[10] M. P. Sears and L. J. D. Frink, *A new efficient method for density functional theory calculations of inhomogeneous fluids*, J. Comp. Phys., 190 (2003), pp. 184–200.

# STOKES FLOW WITH CAPILLARY FORCES USING SUNDANCE

JOHN W. FETTIG[*] AND S. SCOTT COLLIS[†]

**Abstract.** We simulate and analyze the flow of a solidifying fluid in a crack as it would occur in proposed self healing composites [17]. The specific system we consider is motivated by ongoing experiments using a monomer healing agent that undergoes a ring opening metathesis polymerization initiated by Grubbs catalyst [9]. In this case, the monomer healing agent flows into a crack bringing it into contact with a catalyst that induces polymerization. We have designed a simulation tool to understand and optimize the coupled micron-scale fluid mechanics, the advection-diffusion of the healing agent, and the solidification. This simulation tool uses a phase-field like model for the changing viscosity, and is capable of modeling surface tension as well as moving contact lines. The simulator was initially developed in Fortran 90 and then transitioned to the current implementation in C++ using Sundance.

**1. Introduction.** The concept of a composite material that is capable of responding autonomically to damage was demonstrated by White *et al.* [17] with a polymer material capable of recovering 75% toughness in response to a crack-initiated heal event. This one-time healing system is technologically promising, especially if it can lead to a material capable of repeated, or continuous, self-healing. This goal of repeated or continuous healing motivated recent efforts to embed a microvascular network, similar to those in biological systems, in the material. Like a circulatory system, such a network is capable of continuous delivery of a healing agent. Our investigation is motivated by a specific system in which Grubbs' catalyst induces dicyclopentadiene (DCPD) monomer to polymerize and heal the crack [9, 10]. The monomer healing agent flows through microvascular channels which run in networks through the material which is embedded with catalyst particles.

The hydrodynamics of such a polymerizing flow plays a key role in the determining how such a system will respond to cracks. The design of a self-limiting reaction, which both heals regions of damage while leaving intact a delivery system to heal future damage, is important for the longevity of such a system. Current systems built with catalyst-embedded polymers with microvasculatures have been successful, but have also shown greater sensitivity of healing effectiveness to system parameters than expected [16].

In this paper, we present a simulation model of such a self-healing system. This simulation model is based on an Eulerian-frame phase-field like solve for viscosity, coupled to an advection-diffusion solver. An additional phase-field provides the capability of modeling multiple fluids with surface tension and dynamic contact line.

The layout of this paper is as follows. In Section 2 we give a brief summary of the governing equations for our model. Then, in Section 3 we summarize the motivation and implementation of our numerical algorithm, as well as some notes on the implementation using Sundance. We then show results for some sample problems in Section 4, which demonstrate the capabilities of the code.

## 2. Governing Equations.

### 2.1. Stokes Flow.

The polymerization's effect on the flow of the monomer is approximated by a sharp rise in the viscosity with a modification to a formula proposed by Roller [8]. Experimental measurements of tertiary amine catalyzed dicyandiamide cured epoxy systems suggest that viscosity rises according to

$$\frac{d\mu(\mathbf{x},t)}{dt} = \mu_o c e^{c\tau} H(\phi(\mathbf{x}) - \phi_{\min}), \tag{2.1}$$

[*]University of Illinois at Urbana-Champaign, jfettig@uiuc.edu
[†]Sandia National Laboratories, sscoll@sandia.gov

where $\mu_o$ is the viscosity of the monomer healing agent and $c$ is a reaction rate parameter. This time $\tau$ is a modification of Roller's formula to account for the time a material point is in contact with catalyst:

$$\tau = \int_0^t H(\phi(\mathbf{x}) - \phi_{\min})\, dt$$

where $H$ is the Heaviside step function, $\phi(\mathbf{x}(t))$ is the catalyst concentration for a material point $\mathbf{x}$, and $\phi_{\min}$ is a regularization parameter set to be 0.01 of the initial peak concentration. For high enough viscosities the fluid is effectively solid.

An equivalent way of writing this, which allows for the use of Eulerian-frame methods, is

$$\frac{d\mu(\mathbf{x}, t)}{dt} = \tilde{c}\mu$$

where $\tilde{c} = \mu_o c H(\phi(\mathbf{x}) - \phi_{\min})$. Using this equation, we can model the viscosity rise using a phase-field like approach, where the advection of viscosity is coupled to a reaction term, $\tilde{c}\mu$. It is worth noting here that we can generalize this model to a two-part resin+hardener epoxy system by changing the form of $\tilde{c}$ to $\tilde{c} = 4\mu_o\phi(1-\phi)H(\phi(\mathbf{x}) - \phi_{\min})H(\phi_{\max} - \phi(\mathbf{x}))$

By this model, $1/c$ is the appropriate time scale for the changing viscosity. Using this time scale along with velocity scale $U$, length scale $L$, pressure scale $\mu_o U/L$, and viscosity scale $\mu_o$, we non-dimensionalize time $t^*$, velocity $\mathbf{u}^*$, position $\mathbf{x}^*$, pressure $p^*$, and viscosity $\mu^*$, respectively, as

$$t = ct^*, \qquad \mathbf{x} = \frac{\mathbf{x}^*}{L}, \qquad \mathbf{u} = \frac{\mathbf{u}^*}{U}, \qquad p = \frac{p^* L}{\mu_o U}, \qquad \mu = \frac{\mu^*}{\mu_o}. \tag{2.2}$$

The Navier-Stokes equations become

$$\nabla \cdot \mathbf{u} = 0 \tag{2.3}$$

$$\frac{\rho c L^2}{\mu_o}\frac{\partial u}{\partial t} + \mathrm{Re}(\mathbf{u} \cdot \nabla)\mathbf{u} = -\nabla p + \nabla \cdot (\mu \nabla \mathbf{u}), \tag{2.4}$$

where the Reynolds number is $\mathrm{Re} = \frac{\rho U L}{\mu_o}$. The systems of interest consist of very small (~0.1mm) channels with a viscous fluid flowing at slow speeds. Thus, the Reynolds number is small enough to neglect the inertia term in (2.4). The more subtle assumption is that $c$ is comparable to or smaller than the time scale $U/L$, which is true for the proposed DCPD-Grubbs' catalyst system. In this case, the time term's coefficient will be comparable to or smaller than the Reynolds number. So in the limit of zero Reynolds number, we have

$$\nabla \cdot \mathbf{u} = 0 \tag{2.5}$$

$$-\nabla p + \nabla \cdot (\mu \nabla \mathbf{u}) = 0, \tag{2.6}$$

the standard Stokes equations with variable viscosity.

**2.1.1. Boundary Conditions.** Fluid-fluid and fluid-wall interactions are important at the length scales of interest. The no-slip assumption has been shown through MD simulations to not hold on such small scales [6, 7]. There is a slip with linear friction [4] at these scales. This is described by the Navier boundary condition:

$$\beta \mathbf{u} \cdot \mathbf{t} = -\mathbf{t} \cdot \tau \cdot \mathbf{n} \tag{2.7}$$

Here, $\beta$ is a friction coefficient, $\tau = \nu\nabla\mathbf{u} - p\mathbf{I}$ is the stress tensor, and $\mathbf{t}$ and $\mathbf{n}$ are the wall tangent and normal vectors, respectively. By combining this with a penetration condition

$$\mathbf{u}\cdot\mathbf{n} = -\alpha\mathbf{n}\cdot\tau\cdot\mathbf{n}, \tag{2.8}$$

we can effectively replace the Stokes flux boundary condition

$$\tau\cdot\mathbf{n} = \mathbf{s}. \tag{2.9}$$

This has been shown to be valid away from the contact line. At the contact line we have almost complete slip, and the slip velocity is proportional to the deviation from the equilibrium contact angle. This slip is being driven by uncompensated Young's stress. To account for this, the boundary condition becomes

$$\beta\mathbf{u}\cdot\mathbf{t} = -\mathbf{t}\cdot\tau\cdot\mathbf{n} + \gamma(\cos\theta_s - \cos\theta) \tag{2.10}$$

In the context of the phase-field method, we can effectively have (2.7) away from the contact line and (2.10) near the contact line by using the gradient of the phase-field $\psi$:

$$\beta\mathbf{u}\cdot\mathbf{t} = -\mathbf{t}\cdot\tau\cdot\mathbf{n} + \gamma\left(\cos\theta_s - \frac{\nabla\psi\cdot\mathbf{n}}{|\nabla\psi|}\right)\nabla\psi\cdot\mathbf{t} \tag{2.11}$$

This is a form of the generalized Navier boundary condition (GNBC) [6].

**2.2. Advection-Diffusion.** The viscosity depends on the concentration of catalyst $\phi$, which is in turn governed by an advection-diffusion equation

$$\frac{\partial\phi}{\partial t} = -(\mathbf{u}\cdot\nabla)\,\phi + \nabla\cdot(D\nabla\phi)\,. \tag{2.12}$$

We assume that the diffusion parameter $D$ is related to the viscosity by the Stokes-Einstein relation, which is parameterized by the Schmidt number as

$$D = \frac{\rho}{\mathrm{S}c\,\mu}. \tag{2.13}$$

This model is an accepted approximation and is appropriate for the phenomenological investigation we report here. The algorithm admits more general constitutive models and can be refined to match any particular material.

**2.3. Phase-field.** We introduce a phase-field $\psi$ in order to model free surfaces and two fluid flows, where free surface effects such as surface tension and dynamic contact line are important. The phase-field is passively advected by the flow field, via

$$\frac{D\psi}{Dt} = 0.$$

$\psi$ is initialized to be a "smoothed" Heaviside function, with 0 representing fluid 0 and 1 representing fluid 1. The "smoothed" function in this case is

$$\psi_o = \frac{1 + \tanh\left(\frac{\mathbf{n}}{W}\right)}{2}$$

where $\mathbf{n}$ is the distance in the normal direction from the interface and $W$ is a measure of the width of the interface. Typically $W$ will be proportional to the diameter of the elements close to the interface.

The equilibrium contact angle can be enforced using a boundary condition to the Stokes flow equation, as described in §2.1.1. We also add a surface tension term to the Stokes equation:

$$\mathbf{f}_\sigma = \sigma\kappa(\mathbf{x})\delta\left[\mathbf{n}\cdot(\mathbf{x}-\mathbf{x}_s)\right]\mathbf{n}$$

where $\kappa(\mathbf{x})$ is the curvature of the interface and $\mathbf{x}_s$ is the location of the interface. $\kappa$ can be stated in terms of the normalized gradient of the phase-field as

$$\kappa(\mathbf{x}) = -\nabla\cdot\left(\frac{\nabla\psi}{|\nabla\psi|}\right)$$

and we approximate $\delta\left[\mathbf{n}\cdot(\mathbf{x}-\mathbf{x}_s)\right]\mathbf{n}$ as

$$\delta\left[\mathbf{n}\cdot(\mathbf{x}-\mathbf{x}_s)\right]\mathbf{n} = \nabla\psi.$$

Since the phase-field approximation sharpens with decreasing element diameter ($h$) near the interface, in the limit that $h \to 0$ we recover the delta function [1].

**3. Numerical Method.** We solve the Stokes flow equation using Taylor-Hood finite elements. The viscosity $\mu$, and the phase field $\psi$ are calculated on linear elements. The catalyst concentration $\phi$ is calculated on quadratic elements.

**3.1. Stabilization.** The advection-reaction equation for viscosity and the advection equation for the phase-field are stabilized using Streamline Upwind/Petrov-Galerkin [2]. The choice of stabilization parameter $\tau$ is a trade-off between the popular choices for the advection-dominated case and the transient-dominated case [12, 15]:

$$\tau = \left[\left(\frac{2|\mathbf{u}|}{h}\right)^2 + \left(\frac{2}{\Delta t}\right)^2\right]^{-1/2}$$

This trade-off is essential in the viscosity calculation where it is possible for the interface between high and low viscosity regions to have high transients but low advection and vice-versa.

**3.2. Hybrid Implicit-Explicit Time stepping.** All of the equations are advanced using a hybrid implicit-explicit time stepping scheme which appears in the appendix of [13]. The need for a hybrid approach arises because of non-constant diffusivities in the advection-diffusion equation, and a non-constant diffusion-like term in the stabilized advection equation. A fully explicit scheme would have severe time step restrictions, whereas a fully implicit scheme is challenging due to the non-constant diffusivities.

We proceed by writing the advection-diffusion equations as

$$\frac{\partial\phi}{\partial t} = L(\phi) + N(\phi) = R(\phi)$$

All of the terms which we want to treat with an implicit method should appear in $L$ and all of the terms treated explicitly appear in $N$. The term we are trying to treat implicitly is the diffusion term. Since it is non-constant, we further split it into a constant (but large) term and a changing (but small) term:

$$\nabla\cdot(D(\mathbf{x},t)\nabla\phi) = \nabla\cdot(D(\mathbf{x},t_o)\nabla\phi) + \nabla\cdot\left[(D(\mathbf{x},t) - D(\mathbf{x},t_o))\nabla\phi\right]$$

Now we treat the first term with a Crank-Nicolson like scheme and we couple that with an explicit third order Runge-Kutta scheme to treat the second term and the rest of the advection-diffusion equation. This splitting changes our explicit time step restriction from $\Delta t \leq \frac{\Delta x^2}{2D}$ to $\Delta t \leq \frac{\Delta x}{2}$ since $D(\mathbf{x}, t) - D(\mathbf{x}, t_o) \sim \Delta t \sim \Delta x$.

Completely analogous splitting is done for the stabilization terms in the SU/PG stabilized advection operator, where a diffusion-like operator appears.

**3.3. Gradient and curvature reconstruction.** Since we are computing the phase-field on linear elements, the curvature, which is a second derivative, requires some special treatment. We reconstruct the curvature using a sequence of minimization problems on the functionals $\mathbf{g}$ and $\kappa$:

$$I_g = \int_\omega \frac{1}{2}(\mathbf{g} - \nabla\psi)^2 \, d\Omega$$
$$I_\kappa = \int_\omega \frac{1}{2}\left[\kappa - \nabla \cdot \left(\frac{\mathbf{g}}{|\mathbf{g}|}\right)\right]^2 \, d\Omega$$

This variational problem is solved on linear elements, giving us a projection of the gradient and of the curvature onto linear elements. The curvature calculated in this manner is only accurate near the interface, where $|\nabla\psi| \gg 0$, but since $\kappa$ appears in the governing equations multiplied by $|\nabla\psi|$, this is not an issue. In future work it may be advisable to solve this minimization problem for $\kappa|\nabla\psi|$ instead of $\kappa$.

**3.4. Sharp interface tracking with Phase-fields.** The phase-field is initialized to a hyperbolic tangent profile, and ideally this should remain a hyperbolic tangent for all time, regardless of the advection operator and topology changes. In practice this does not occur because of the finite width of the interface and the possibility for gradients to exist in the velocity field across this interface. Therefore it becomes desirable to maintain the fixed width of the hyperbolic tangent profile over all time through a correction to the phase-field advection equation. This phenomenon is more often associated with the level-set method, where maintaining a signed distance function is important and requires significant effort [11].

To accomplish this, we use a correction term introduced by Beckermann *et al.* [14] to the advection operator. It is based on the kernel function

$$\psi_o = \frac{1 + \tanh\left(\frac{\mathbf{n}}{W}\right)}{2}$$

By substituting this kernel function into the equation for curvature, we get

$$\kappa = -\frac{1}{|\nabla\psi|}\left[\nabla^2\psi + \frac{4\psi(2\psi - 1)(\psi - 1)}{W^2}\right]$$

We use this to correct the phase field by adding a weighted difference of this with the actual curvature, calculated in §3.3, to the phase-field advection equation:

$$\begin{aligned}\frac{D\psi}{Dt} &= b(\kappa|\nabla\psi| - \kappa|\nabla\psi|) \\ &= b\left(-\nabla^2\psi - \frac{4\psi(2\psi - 1)(\psi - 1)}{W^2} - \kappa|\nabla\psi|\right)\end{aligned}$$

Again, we have a diffusion operator which is treated implicitly in our hybrid time-stepping scheme. $b$ is a purely numerical parameter which is chosen to be small.

At the time of this writing, this correction term is disabled since it seems the errors introduced by "stepping up" the curvature result in an over-diffuse correction term here. Alternative approaches to calculating this correction term, as well as the surface tension term in the Stokes equation, need to be investigated. Integration by parts of both of these terms may possibly eliminate the overly diffuse nature.

**3.5. Notes on implementation in Sundance.** The implementation of all of these governing equations in Sundance [5] is relatively straightforward. In order to re-use the linear solve operators over the course of the time stepping iterations, we have created objects which contain each of the governing equations separately. As each of the variable quantities in the governing equations are updated, the corresponding pointers in the objects are switched to point to the updated values. In this way, we construct the integrands and linear operators only once, although the integrals are re-evaluated at each time step.

Sundance proved to be a valuable tool in the implementation of the weak forms of these equations. It allowed for rapid development of the simulator, and immediately provides parallel support as well as access to Trilinos [3] solvers through either the Trilinos Solver Framework (TSF), or through the Stratimikos interface. In both cases, the solver can be changed by modification of an XML file.

## 4. Results on sample problems.

**4.1. Capillary flow.** For this problem, we model flow in a 2D rectangular channel with an equilibrium contact angle of $\pi/4$ as a demonstration of the ability to model dynamic contact lines. The viscosity is constant throughout the simulation. The phase field is initialized as

$$\psi_o = \frac{1 + \tanh\left(\frac{5-x}{W}\right)}{2}$$

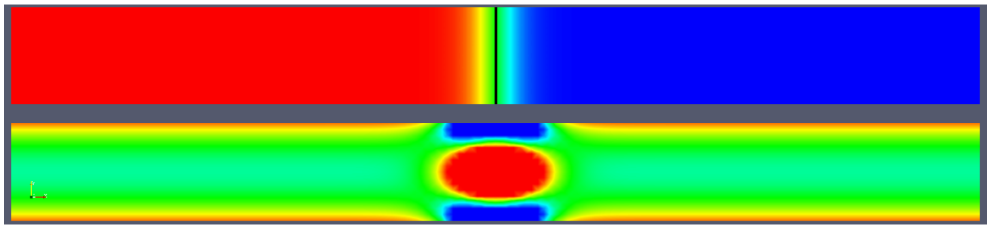
so that the initial contact angle is $\pi/2$, and the interface between the two fluids is centered at $x = 5$. This deviation from equilibrium causes a flow driven by capillary forces. Surface tension is also present.

Results are shown in Figure 4.1. The $\psi = 0.5$ contour line is overlaid on the plot of the phase-field, and the $u$ component of velocity is shown. The system is farthest from equilibrium at $t = t_o$, and so the velocity is strongest there. The fluid exhibits near total slip at the wall at this time. The system moves towards the equilibrium contact angle and so the velocity slows until it reaches an equilibrium between the contact angle stress and the surface tension force.

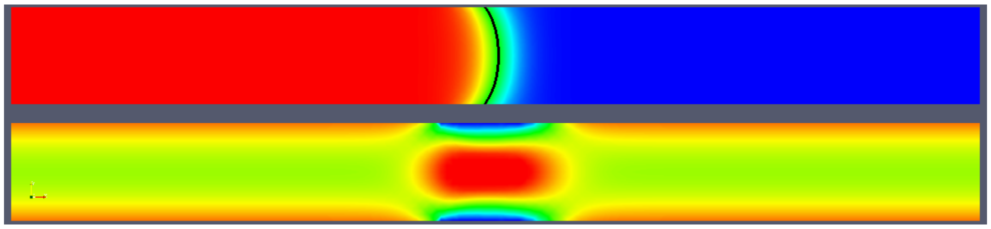
**4.2. Two part system with free surface.** For this problem, we model the two part system shown in Figure 4.2. Two embedded networks are flowing through the material: one contains resin, the other contains hardener. A coating is placed on the top of the specimen to seal the networks. When the specimen is placed into a 4-point bend, the coating cracks allowing the two networks to release fluid into the crack plane. When the two fluids contact, they react causing polymerization.

Figure 4.3 depicts a simulation of two channels, one with resin and the other with hardener. There is a pressure forcing the fluids to eject into the crack plane, and we initialize $\phi$ to be a Heaviside function with $\phi = 0.5$ in between the two channels. We see hardening at the interface of the two fluids, but this hardening is localized. It does not impede the flow.

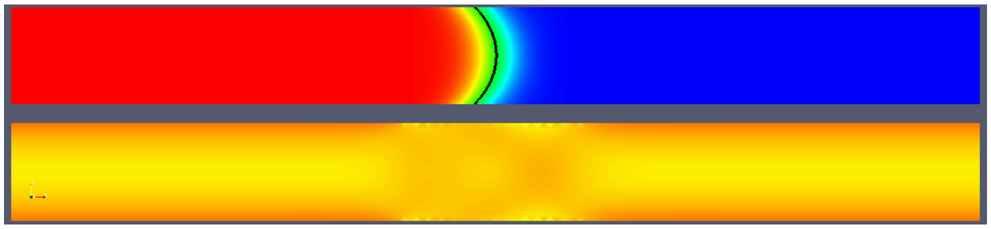
**5. Conclusions.** A parallel simulator has been developed using Sundance for simulation of microvascular self-healing flows. This simulator is capable of modeling free surface flows, dynamic contact lines, and polymerizing flow in small scales. For future work, we will use this code to optimize the physically tunable parameters.



(a) $t = t_o$

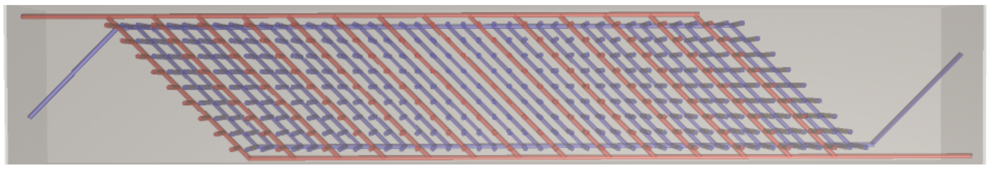


(b) $t = t_1$

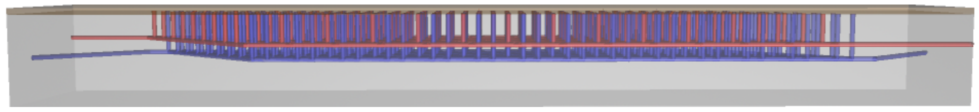


(c) $t = t_2$

FIG. 4.1. *The phase-field, $\psi$, with contour $\psi = 0.5$ overlaid, and the u component of velocity at three different time steps. Red indicates $\psi = 1$ and blue indicates $\psi = 0$. For u, red indicates greatest velocity and blue indicates smallest velocity.*

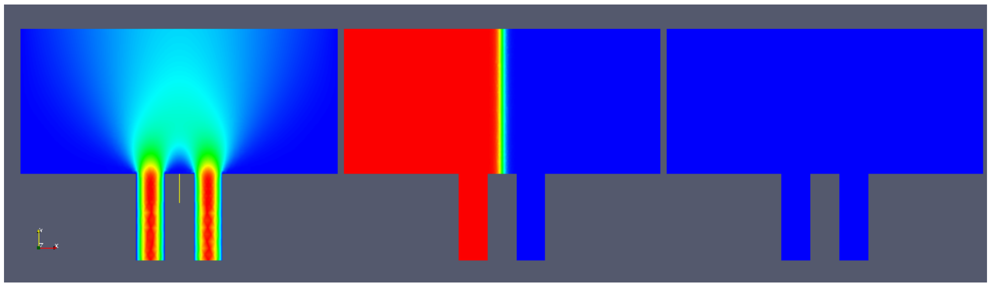


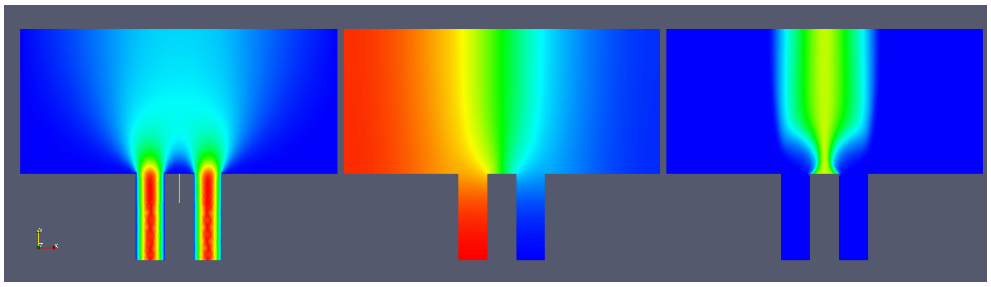
(a) Top view



(b) Side view

FIG. 4.2. *The model 2 part epoxy system. The blue network contains resin and the red network contains hardener. These two networks are completely segregated, and only mix in the crack plane (on the top surface in this schematic).*


(a) $t = t_o$


(b) $t = t_1$

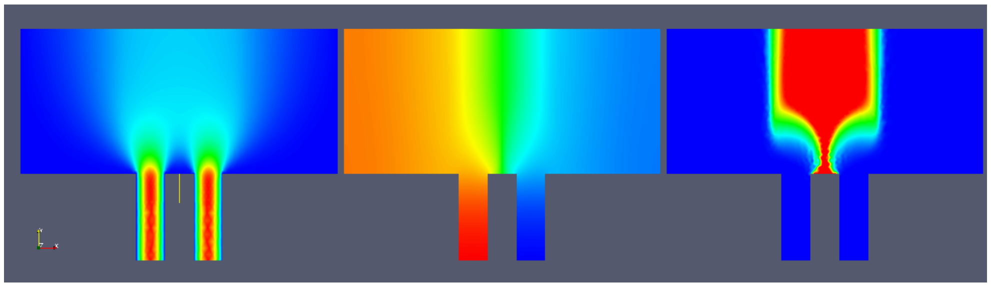

(c) $t = t_2$

FIG. 4.3. *The v velocity component, the resin-hardener ratio $\phi$, the u component of velocity at three different time steps. and the viscosity $\mu$ at three different time steps. For v ($\mu$), red indicates greatest velocity (viscosity) and blue indicates smallest velocity (viscosity). For $\phi$, red indicates $\phi = 1$ and blue indicates $\phi = 0$.*

**6. Acknowledgments.** The authors greatly appreciate the help and insight of Ross Bartlett, Kevin Long, Judy Hill, and Bart van Bloemen Waanders on all things Sundance.

## REFERENCES

[1] J. U. Brackbill, D. B. Kothe, and C. Zemach, *A continuum method for modeling surface tension*, Journal of Computational Physics, (1992), pp. 335–354.

[2] N. Brooks and J. R. Hughes, *Streamline upwind/petrov-galerkin formulations for convection dominated flows with particular emphasis on the incompressible navier-stokes equations*, Computer Methods in Applied Mechanics and Engineering, (1982), pp. 199–259.

[3] M. Heroux, R. Bartlett, V. H. R. Hoekstra, J. Hu, T. Kolda, R. Lehoucq, K. Long, R. Pawlowski, E. Phipps, A. Salinger, H. Thornquist, R. Tuminaro, J. Willenbring, and A. Williams, *An Overview of Trilinos*, Tech. Report SAND2003-2927, Sandia National Laboratories, 2003.

[4] V. John, *Slip with friction and penetration with resistance boundary conditions for the navier-stokes equations – numerical tests and aspects of the implementation*, Journal of Computational and Applied Mathematics, (2002), pp. 287–300.

[5] K. Long, *Sundance User's Manual*, Tech. Report SAND2004-4793, Sandia National Laboratories, 2004.

[6] T. Qian, X. Wang, and P. Sheng, *Molecular scale contact line hydrodynamics of immiscible flows*, Physical Review E, (2003).

[7] W. Ren and W. E, *Boundary conditions for the moving contact line*, Physics of Fluids, (2007).

[8] M. B. Roller, *Characterization of the time-temperature-viscosity behavior of curing b-staged epoxy resin*, Polymer Engineering and Science, 15 (1975).

[9] J. Rule and J. Moore, *Romp reactivity of endo- and exo-dicyclopentadiene*, Macromolecules, 35 (2002), pp. 7878–7882.

[10] J. Rule, N. Sottos, S. White, and J. Moore, *The chemistry of self-healing polymers*, Education in Chemistry, 42 (2005), pp. 130–132.

[11] J. A. Sethian, *Level set methods and fast marching methods: evolving interfaces in computational geometry, fluid mechanics, computer vision, and materials science*, Cambridge University Press, 1999.

[12] F. Shakib, *Finite element analysis of the compressible Euler and Navier-Stokes equations*, PhD thesis, Stanford University, 1989.

[13] P. R. Spalart, R. D. Moser, and M. M. Rogers, *Spectral methods for the navier-stokes equations with one infinite and two periodic directions*, Journal of Computational Physics, (1991), pp. 297–324.

[14] Y. Sun and C. Beckermann, *Sharp interface tracking using the phase-field equation*, Journal of Computational Physics, (2007), pp. 626–653.

[15] T. E. Tezduyar and Y. Osawa, *Finite element stabilization parameters computed from element matrices and vectors*, Computer Methods in Applied Mechanics and Engineering, (2000), pp. 411–430.

[16] K. Toohey. personal communication, 2006.

[17] S. White, N. Sottos, P. Geubelle, J. Moore, M. Kessler, S. Sriram, E. Brown, and S. Viswanathan, *Autonomic healing of polymer composites*, Nature, (2001), pp. 794–797.

# CONTINUATION FOR ATOMIC AND MOLECULAR FLUIDS USING LOCA

KELLY I. DICKSON[*], ANDREW G. SALINGER[†], B. MONTGOMERY PETTITT[‡], MARCELO MARUCHO[§], AND C.T. KELLEY[¶]

**Abstract.** We wish to obtain accurate structural and thermodynamic properties of both atomic and molecular fluids. Current equations used to describe such quantities have significant shortcomings. In [8], Marucho and Pettitt propose an improvement upon the existing theory. We present a computational approach to solving this new theory and enable simulations of fluid properties as a function of fluid densities. This analysis can be accomplished through a continuation study. We present continuation results for atomic fluids using the software package LOCA in Trilinos, a Sandia National Laboratories solver framework. Further, we discuss how to approach a similar continuation study for molecular fluids.

**1. Introduction.** Many physical systems that surround us are best described on an atomic or molecular level. In fact, American physicist Richard Feynman said in 1963, "Everything that living things do can be understood in terms of the jiggling and wiggling of atoms." One way of analyzing molecular properties and interactions is known as integral equation theory. Often one is interested in molecular structure and thermodynamic properties of a system at thermodynamic equilibrium (time independent). Integral equation theorists seek to derive governing equations that yield more accurate structural and thermodynamic information with less computation than, say, simulation (molecular dynamics). Here we focus on improving integral equation theory to recover molecular structure and thermodynamic properties through analyzing *closure equations*. In particular, we discuss a new closure equation recently developed [8] by Marcelo Marucho and B.M. Pettitt of the University of Houston's Department of Chemistry.

In the next section, we will explain the terms "structure" and "thermodynamic properties" by considering *atomic* and *molecular* fluids. We will introduce integral equation theory and closure equations by first discussing atomic fluids and then later extending the theory to molecular fluids. Along the way, we will point out some crucial deficiencies in existing equations found in the literature. This will motivate the need for improvements in integral equation theory, and in particular, the one presented in [8]. We then relate the analysis of the molecular theory to numerical continuation. Specifically, we introduce the continuation problem of interest within the context of fluids. We discuss a new implementation for this continuation problem using software developed at Sandia National Laboratories. Finally, we present some current results and the future aims of this project.

**2. Introduction to Atomic and Molecular Fluids.** This section describes some properties of fluids and why they are important. Later, we will focus on how to obtain these properties.

As mentioned earlier, integral equation theory is a route to uncovering structural and thermodynamic properties of fluids. The two general categorizations of fluids we will discuss are *atomic fluids* and *molecular fluids*. Atomic fluids are ones that are comprised of only single atoms while molecular fluids are made up of molecules (groups of atoms bonded together). In particular, we will consider homogeneous atomic and molecular fluids, i.e. fluids that only contain one kind of atom or one kind of molecule. For molecular fluids, we focus on

[*]North Carolina State University, kidickso@ncsu.edu
[†]Sandia National Laboratories, agsalin@sandia.gov
[‡]University of Houston, pettitt@uh.edu
[§]University of Houston, marucho@kitten.chem.uh.edu
[¶]North Carolina State University, tim_kelley@ncsu.edu

diatomic (molecules made up of only two bonded atoms) fluids. When convenient, we will use the term "particle" to refer to either an atom or molecule.

Integral equation theory is useful for finding structural and thermodynamic properties of particle fluids at equilibrium. In particular, it is useful for finding *pair correlation functions* which we will denote $g(r)$. Here, $r$ represents the distance between two particles in a fluid. The pair correlation function, $g(r)$, is the relative probability of finding a particle at a distance, $r$, from the reference particle (at $r = 0$) compared to that same probability for an uncorrelated fluid. Figure 2 is an example of what a pair correlation function may look like. A general characterization of a correlation function is that there is an exclusion region at small $r$ where the particles are prohibited from overlapping, followed by an oscillatory region where the particles pack around the reference particle which then decays to an uncorrelated fluid. The way particles in a fluid are correlated is known as the "structure" of a fluid.

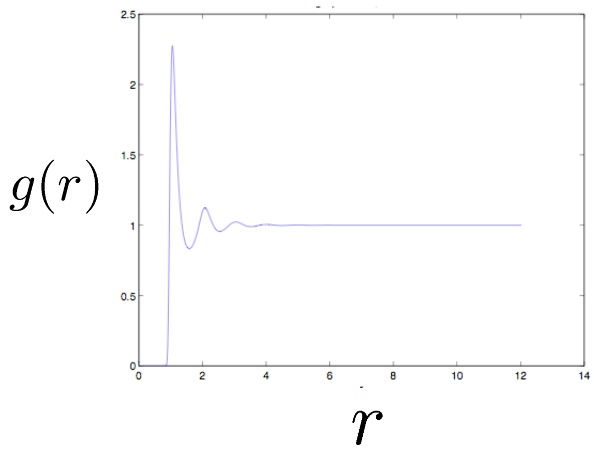


FIG. 2.1. *Pair correlation function $g(r)$ for a particular density and temperature*

Additionally, it is desired to predict thermodynamic properties associated with a fluid. We can relate properties of atoms and molecules on a microscopic level to properties of materials that we observe in everyday life on a macroscopic level through thermodynamics. Thermodynamics are the effects of changes in conditions such as temperature, pressure on important fluid properties such as compressibility, density, excess adsorption, etc. of a physical system. Many thermodynamic quantities are functions of the pair correlation function, $g(r)$. That is, the structure of $g(r)$ gives rise to many thermodynamic properties of interest.

To understand the equations that describe particle fluids, we need to discuss a few more types of correlation functions. The first is called the *direct correlation function* which we will denote $c(r)$. The function $c(r)$ represents the direct correlation between two particles in a fluid. This is opposed to $t(r)$, called the *indirect correlation function*, which also describes the correlation between two particles in a fluid, but takes into consideration that the two particles are surrounded by other particles of the same type. That is, $t(r)$ measures implicit correlations due to the presence of other particles in the fluid. Then the *total correlation function*, denoted $h(r)$, is the direct correlation plus indirect correlation. In other words, $h(r) = c(r) + t(r)$. Finally, the pair correlation function of interest is given by $g(r) = h(r) + 1$.

With these definitions, we will now introduce the underlying integral equation for atomic fluids studied in this work.

**3. Atomic Fluids.** One theory that has been commonly used to predict structural, and therefore thermodynamic, properties of atomic fluids is the Ornstein-Zernike (OZ) equation [5, 2].

**3.1. Integral Equation Theory for Atomic Fluids.** The OZ equation is an integral equation with two unknown fields. These are the total correlation function $h(r)$ and the direct correlation function $c(r)$, as defined in §2. The OZ equation is given by

$$h(r) = c(r) + \rho(h * c)(r) \tag{3.1}$$

where

$$(h * c)(r) = \int_{\Re^3} c(\|\mathbf{r} - \mathbf{r}'\|)h(\|\mathbf{r}'\|)d\mathbf{r}'.$$

Here,

- $\mathbf{r} \in \Re^3$,
- $r = \|\mathbf{r}\|$ is the distance of $\mathbf{r}$ from the origin,
- $\rho$ is density, and
- $h(r), c(r) \in C[0, \infty)$ are the unknowns.

In order to resolve (3.1) and recover the pair correlation function $g(r)$, we need to solve for the two unknowns $h(r)$ and $c(r)$. To do this, we need one more equation that relates the two unknowns in order to close the system. This equation is called a *closure equation*, or better yet, a *closure approximation*. Most closure equations involve an infinite sum of multidimensional integrals that are difficult to evaluate, aside from the fact there are infinitely many. Thus closure equations must be approximated, often by truncating this infinite sum in some way. While there are some cases in which common closure approximations yield an accurate pair correlation function, they are notorious for displaying thermodynamic inconsistencies. This occurs when two different formulations for a thermodynamic quantity produce two different answers when they should be the same. This is exactly the kind of inconsistency we wish to avoid since, as earlier discussed, the accurate prediction of thermodynamic properties is the purpose of simulating the system.

**3.2. Closure Equations for Atomic Fluids.** Here we define two common yet insufficient closure equations used to find the the solutions $h(r)$ and $c(r)$ to the OZ equation (3.1). Then we propose a new closure approximation developed in [8] for atomic fluids.

The hypernetted chain equation [5, 2] (HNC) is one popular closure approximation used to resolve (3.1), however it can produce inaccurate pair correlation functions and/or thermodynamic inconsistencies. The HNC approximation is given by

$$\exp(-u(r)/(TK_B) + h(r) - c(r)) - h(r) - 1 = 0, \qquad 0 \le r \le \infty.$$

In this equation, $u(r)$ is called the pair potential between particles. Throughout this paper, we will consider only the Lennard-Jones potential which describes both an attractive and repulsive interaction between particles. It is given by

$$u(r) = 4\epsilon\left(\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6}\right). \tag{3.2}$$

Additionally,

- $T$ is absolute temperature (Kelvin),
- $K_B$ is Boltzmann's constant (kcal/Kelvin),
- $\epsilon$ is the depth of the attractive interaction (kcal/mol), and
- $\sigma$ is the width of the repulsive core (Angstroms).

A common alternative to the HNC closure is the Percus-Yevick [5, 2] (PY) closure given by

$$\exp(-\beta u(r) + ln(h(r) - c(r) + 1)) - h(r) - 1 = 0.$$

Due to the structural and thermodynamic inconsistencies resulting from the use of these closure equations, Marcelo Marucho and B.M. Pettitt of the University of Houston have developed a new closure equation that "interpolates" between the HNC and PY closures [8]. The new interpolating closure for atomic fluids, which we will denote IC, is given by

$$\exp(-\beta u(r))\left(-a + (a+1)\exp\left(\frac{h(r) - c(r)}{a+1}\right)\right) - h(r) - 1 = 0 \tag{3.3}$$

where the additional parameter $a$ is chosen to minimize excess chemical potential (free energy), $\mu(h(r), c(r), a)$ which we will define momentarily. Note that when $a = 0$, IC reduces to HNC. Additionally, when $a \to \infty$, the IC equation becomes the PY equation. Thus we can think of IC as considering the family of closures, parameterized by $a$, formed by "interpolating" between HNC and PY and choosing the one that minimizes free energy. The idea of minimizing free energy comes from first principles in physics. Note that, in comparison to the HNC and PY closures, this closure adds an optimization condition to the governing equations.

The excess chemical potential $\mu$ is the key thermodynamic quantity in this derivation since it is minimized when the system is in thermodynamic equilibrium. It can be approximated with the formula [8]

$$\beta\mu(h(r), c(r), a) \approx -\rho \int \{h(r) - (h(r) - c(r))\mathcal{I}(h(r), c(r), a)\} d\mathbf{r} \tag{3.4}$$

where

$$\begin{aligned}&\mathcal{I}(h(r), c(r), a) = \{[h(r) + 1]ln[y(r)/a] + ln[a] +\\ &[h(r)(a+1)/(h(r) - c(r))]Re[li_2(y(r)/a + 1) - li_2((a+1)/a)]\}/(h(r) - c(r)).\end{aligned}$$

Here, $li_2$ denotes the dilogarithm function [1] and $y(r) = -a + (a+1)\exp[(h(r) - c(r))/(a+1)]$.

Marucho and Pettitt implement IC together with OZ for atomic fluids in [8]. Their findings result in more accurate structural and thermodynamic properties than the HNC closure yields. Notice at this time that the OZ equation (and IC) depends on the bulk fluid density $\rho$. It is possible that one may wish to obtain accurate information for a fluid with a particular density. In this case, one can plug in the density value of interest into the OZ equation and use IC to obtain the pair correlation function and other information about the system. However, often one is interested in how the structure of the fluid changes as the density changes. The computational process for understanding a system's solution behavior as a parameter, such as density, varies is known as *numerical continuation* [6]. We now briefly discuss numerical continuation and its application to the OZ and IC equations.

**3.3. Numerical Continuation.** As previously mentioned, in order to solve the OZ equation together with the IC closure relation, one must specify a parameter value for density, $\rho$. However it is possible to get a sense of how solution behavior (correlation functions) changes as density values vary. This process is called numerical continuation [6].

Consider a set of nonlinear, parameter dependent equations of the form

$$G(u, \lambda) = 0 \tag{3.5}$$

where $u \in \Re^N$ is the unknown and $\lambda \in \Re$ is a real number parameter (although in general, there may be more than one parameter). The idea of numerical continuation is to find solutions $u$ corresponding to various values of $\lambda$ and to investigate the solution behavior as $\lambda$ varies. In particular, one is often interested in detecting solution paths that undergo bifurcations. In molecular theory, the common bifurcations encountered correspond to phase transitions[9]. In the context of the OZ equation, we can think of correlation functions $h(r)$ and $c(r)$ as the unknown $u$ and the parameter $\rho$ as $\lambda$. $G$ is then like the OZ equation paired with the IC closure approximation. We would like to understand what happens to the solutions $h(r)$ and $c(r)$ as the parameter $\rho$ varies.

There are many established numerical techniques and software dedicated to solving numerical continuation problems like the one just described. Trilinos (overseen by Mike Heroux, 1414) is a collection of open source software packages written in C++, each one designed by a Sandia National Laboratories development team. The packages may stand alone, but are designed to integrate with one another. NOX (Nonlinear Object-Oriented Solutions) is the nonlinear solver package headed by Roger Pawlowski (1416). Built upon NOX is a package called LOCA (Library of Continuation Algorithms) developed by Eric Phipps and Andrew Salinger (1416). LOCA and NOX are the primary packages responsible for the results presented in this paper. For more on Trilinos including download information, visit `http://trilinos.sandia.gov`.

We now turn our focus to implementing continuation in density for the OZ and IC equations with the help of Trilinos.

**3.4. Continuation for the OZ and IC Equations for Atomic Fluids.** We want to solve the OZ and IC equations and analyze the solutions for various density values. This means that for each density value $\rho$, we must resolve the optimization problem of minimizing the excess chemical potential $\mu(h(r), c(r), a)$ with respect to the interpolation parameter $a$. With each continuation step, in order to minimize (3.4) with respect to $a$, one could simply run a suitable optimization scheme such as a descent or simplex method. Then once the minimizer $a$ is found, it can be plugged into the closure equation to solve for $h(r)$ and $c(r)$ as described in [3]. While this is the most intuitive approach, there may be a sufficiently accurate way to minimize the free energy by simply tweaking the continuation problem a bit.

The idea, proposed in this work is to augment our main equation (3.5) with additional equations so that when the new system is solved, we will approximate the value of $a$ that minimizes $\mu(h(r), c(r), a)$ *and* simultaneously solve the OZ equation paired with IC. Then the continuation problem will be applied to this new system of equations.

We will formulate the optimization problem to require a first order difference approximation to the derivative of $\mu(h(r), c(r), a)$ with respect to $a$ to be small. That is, we wish to find $a$ that makes $d\mu/da = 0$ which signifies that we are at a critical point. Thus if $\mu(h(r), c(r), a)$ behaves nicely (and we assume it does), this will imply we are at a minimum.

The new system is formulated as follows. We arrange (3.1) and (3.3) into the form $G = 0$ as done in [3]. We define $u = (h, c)$ and consider $a$ to be another unknown. Let $x = (u, v, a)$ be the new unknown vector (where $v$ is a vector of the same length as $u$). Then instead of solving $G(u) = 0$, solve

$$F(x) = \begin{pmatrix} G(u,a) \\ G(u+\epsilon v, a+\epsilon) \\ [\mu(u+\epsilon v, a+\epsilon) - \mu(u,a)]/\epsilon \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}.$$

The first two equations specify that there are two nearby points of the form $(u, a)$ on the curve $G = 0$ that are just separated by a perturbation $\epsilon$ in the parameter $a$. The third equation says that the first order approximation to the derivative of $\mu$ at the first point is small, meaning we're at a minimum. Finding the $x$ that satisfies $F(x) = 0$ is the minimization problem. To perform the system analysis, one then applies continuation in $\rho$ on

$$F(x,\rho) = 0. \tag{3.6}$$

We have just described a way to implement continuation for the OZ and IC equations for atomic fluids. While some mathematical analysis needs to be done on this approach, it is expected that one can obtain sufficiently accurate structural and thermodynamic properties of an atomic fluid for various parameter values using this technique. We put these ideas to the test by performing a continuation study in $\rho$ while holding $T = 1.827$ fixed. We truncate the integral in OZ at $L = 9.0$, and are unable to continue past $\rho = .7$ where the continuation run started to stall. In Figure 3.1, we plot $\rho$ against the weighted $L_1$ norm of $h$ (left) and weighted $L_1$ norm of $c$ (right). The weighted $L_1$ norm is given by

$$||z||_1 = (1/N)\sum_{i=1}^{N} |z_i|$$

for a vector $z$ of length $N$. The $L_1$ norm is merely a numerically convenient metric of the solution, yet in analyzing these results from a scientific standpoint, one would likely plot the solutions $h(r)$ and $c(r)$ as a function of some physically meaningful quantity.

In order to verify that these solution branches are behaving as expected, we display a solution profile for both $h(r)$ and $c(r)$ at a particular value of $\rho$, in this case, $\rho \approx .522$ (as indicated by the arrows in Figure 3.1). We were able to match these solution profiles with previously known solutions to OZ. These $h(r)$ and $c(r)$ solution profiles were done with other values of $\rho$, although we only display one here. These results give us reason to believe that solving OZ paired with IC as described in this chapter is a promising approach.

**4. Extension to Molecular Fluids.** So far we have described a way to recover accurate structural and thermodynamic properties of atomic fluids with varying densities using the "interpolating" closure. We now extend this idea to more complicated molecular fluids. First, we present the "analog" of the OZ and IC equations for molecular fluids.

**4.1. Integral Equation Theory for Molecular Fluids.** Integral equation theory for molecular fluids is more complex than the OZ equation for atomic fluids. As mentioned in §2, we will discuss the theory for specifically for diatomic molecular fluids. An integral equation theory for diatomic molecular fluids, and the one considered here, was developed by Dyer, Perkyns, and Pettitt [4] and was found to be an improvement upon existing theories such as PISM (proper interaction site model)[7]. The equations are in matrix-product form and given in Fourier space. The "hat" notation denotes the Fourier transform and $k$ is the independent variable in Fourier space. The molecular fluid equations are given by

$$\hat{\mathbf{H}}(k) = \hat{\mathbf{C}}(k) + [\hat{\mathbf{C}}(k) + \hat{\mathbf{S}}(k)]\bar{\rho}_m[\hat{\mathbf{H}}(k) + \hat{\mathbf{S}}(k)].$$

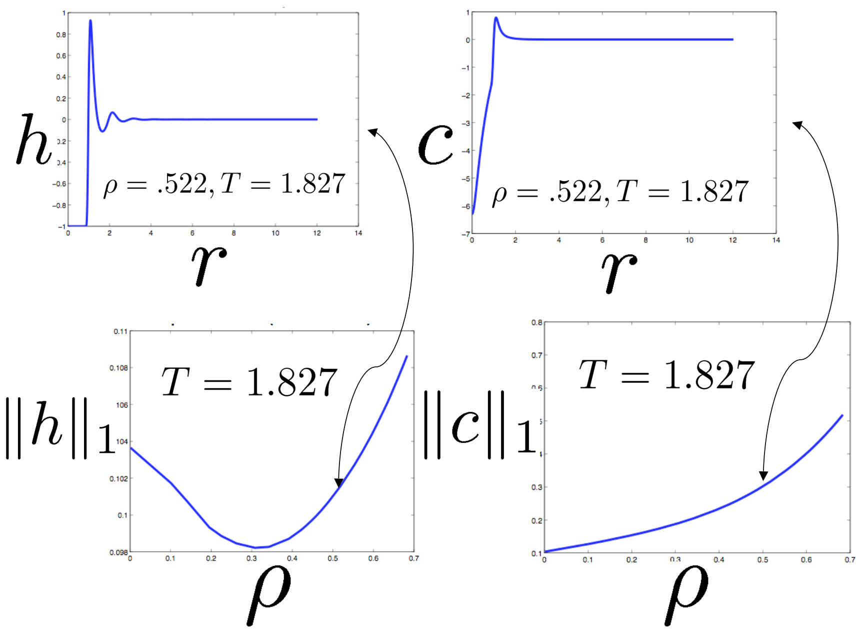


FIG. 3.1. *Solutions $h(r)$ and $c(r)$ as a function of $\rho$ for $T = 1.827$*

Here,

$$\rho_m = \begin{pmatrix} \rho & \eta \\ \eta & 0 \end{pmatrix}, \bar{\rho}_m = \begin{pmatrix} \rho_m & \mathbf{0} \\ \mathbf{0} & \rho_m \end{pmatrix}, \hat{\mathbf{S}}(k) = \begin{pmatrix} \mathbf{0} & s(k) \\ s(k) & \mathbf{0} \end{pmatrix}, s(k) = \eta^{-1} \begin{pmatrix} 0 & 0 \\ 0 & \frac{sin(kL)}{kL} \end{pmatrix},$$

where $\eta$ is called the screened density and $L$ is the bond length between atoms in the molecule. The notations $\rho_m$ and $\bar{\rho}_m$ differentiate the matrix forms of $\rho$ and the regular scalar $\rho$. The unknowns here are the matrix entries of

$$\mathbf{Q}(k) = \begin{pmatrix} \mathbf{Q}_{11}(k) & \mathbf{Q}_{12}(k) \\ \mathbf{Q}_{21}(k) & \mathbf{Q}_{22}(k) \end{pmatrix}, \mathbf{Q}_{\alpha\gamma}(k) = \begin{pmatrix} q^o_{\alpha\gamma}(k) & q^r_{\alpha\gamma}(k) \\ q^l_{\alpha\gamma}(k) & q^b_{\alpha\gamma}(k) \end{pmatrix},$$

where $\mathbf{Q}$ represents either $\mathbf{H}$ or $\mathbf{C}$. The subscript $\alpha = 1, 2$ represents one type of atom in the diatomic molecule and $\beta = 1, 2$ the other so that $\mathbf{H}_{\alpha\gamma}$ or $\mathbf{C}_{\alpha\gamma}$ represents a correlation between atom $\alpha$ in one molecule and atom $\beta$ in another. Finally, the superscripts $o$, $r$, $l$, and $b$ represent further classifications of each atom-atom correlation.

Although in matrix form, we still have two unknowns as in the atomic case, $\mathbf{H}$ and $\mathbf{C}$. Thus we must close the system with a closure equation. As for atomic fluids, the popular HNC and PY closures for molecular fluids result in structural and thermodynamic inconsistencies. Thus we seek a new closure that will remedy these effects, namely, an “interpolating closure” for molecular fluids.

**4.2. Interpolating Closure for Molecular Fluids.** In [8], Marucho and Pettitt introduce the molecular “analog” of the the interpolating closure presented in §3.2. The interpolating closure equations for molecular fluids are given by

$$\begin{aligned}
c^o_{\alpha\gamma}(r) &= -a^o e^{-\beta u(r)} + (1+a^o)e^{[-\beta u(r)+(h^o_{\alpha\gamma}(r)-c^o_{\alpha\gamma}(r))/(1+a^o)]} - 1 - h^o_{\alpha\gamma}(r) + c^o_{\alpha\gamma}(r),\\
c^r_{\alpha\gamma}(r) &= \frac{(1+a^o)}{(1+a^r)}(h^r_{\alpha\gamma}(r) - c^r_{\alpha\gamma}(r))e^{[-\beta u(r)+(h^o_{\alpha\gamma}(r)-c^o_{\alpha\gamma}(r))/(1+a^o)]} - h^r_{\alpha\gamma}(r) + c^r_{\alpha\gamma}(r),\\
c^l_{\alpha\gamma}(r) &= \frac{(1+a^o)}{(a+a^r)}(h^l_{\alpha\gamma}(r) - c^l_{\alpha\gamma}(r))e^{[-\beta u(r)+(h^o_{\alpha\gamma}(r)-c^o_{\alpha\gamma}(r))/(1+a^o)]} - h^l_{\alpha\gamma}(r) + c^l_{\alpha\gamma}(r),\\
c^b_{\alpha\gamma}(r) &= (1+a^o)\left[\frac{(h^b_{\alpha\gamma}(r) - c^b_{\alpha\gamma}(r))}{(1+a^b)} + \frac{(h^r_{\alpha\gamma}(r) - c^r_{\alpha\gamma}(r))(h^l_{\alpha\gamma}(r) - c^l_{\alpha\gamma}(r))}{(1+a^r)^2}\right]\times\\
&\qquad e^{[-\beta u(r)+(h^o_{\alpha\gamma}(r)-c^o_{\alpha\gamma}(r))/(1+a^o)]} - h^b_{\alpha\gamma}(r) + c^b_{\alpha\gamma}(r).
\end{aligned}$$

Notice there are now three interpolation parameters for the molecular case, $a^o$, $a^r$, and $a^b$. Once again, $u(r)$ is the Lennard-Jones pair potential between particles and $\beta$ is the inverse of temperature times Boltzmann's constant. Similar to §3.2, we now must choose the values of $a^o$, $a^r$, and $a^b$ to minimize the approximate excess *molecular* chemical potential given by

$$\begin{aligned}
&\beta\mu(h_{\alpha\gamma}, c_{\alpha\gamma}, a^o, a^r, a^b) \approx -\rho\sum_{\alpha\gamma}\int\{h_{\alpha\gamma} - \mathcal{I}\left[h^o_{\alpha\gamma}(r), c^o_{\alpha\gamma}(r), a^o\right]\times\{h^o_{\alpha\gamma}(r) - c^o_{\alpha\gamma}(r) +\\
&\frac{(1+a^o)\left[h^r_{\alpha\gamma}(r) - c^r_{\alpha\gamma}(r) + h^l_{\alpha\gamma}(r) - c^l_{\alpha\gamma}(r)\right]}{(1+a^r)} + \frac{(1+a^o)\left[h^b_{\alpha\gamma}(r) - c^b_{\alpha\gamma}(r)\right]}{2(1+a^b)}\}\\
&-\frac{h^r_{\alpha\gamma}\left[h^l_{\alpha\gamma}(r) - c^l_{\alpha\gamma}(r)\right]}{(1+a^r)} - \frac{\left[h^r_{\alpha\gamma}(r) + h^l_{\alpha\gamma}(r) + h^b_{\alpha\gamma}(r)\right]\left[h^o_{\alpha\gamma}(r) - c^o_{\alpha\gamma}(r)\right]}{2(1+a^o)}\}d\mathbf{r},
\end{aligned}$$

where $\mathcal{I}(h^o_{\alpha\gamma}(r), c^o_{\alpha\gamma}(r), a^o)$ is obtained by replacing $h(r), c(r)$, and $a$ by $h^o_{\alpha\gamma}(r), c^o_{\alpha\gamma}(r)$, and $a^o$, respectively in the formulation for $\mathcal{I}$ in §3.2.

Now the goal is to solve the matrix equations and molecular interpolating closure just defined for **H** and **C**. This is done in [8] for a few values of density. However, we wish to do a continuation study for molecular fluids as was done in §3.4. Thus the next step is to formulate a problem that will allow a continuation study in $\rho$ while solving the new molecular equations, keeping in mind that we must now minimize the molecular free energy with each continuation step.

**4.3. Current Work for Molecular Fluids.** As done for atomic fluids in §3.4, we can formulate a new continuation problem for molecular fluids that will resolve the optimization problem from §4.2. We can do this by requiring a first order difference approximation to $d\mu/da^o = d\mu/da^r = d\mu/da^b = 0$. Since we now have three interpolation parameters to resolve, the continuation problem in (3.6) grows in size considerably. This idea of minimizing the excess chemical potential for molecular fluids using a first order difference approximation of $\mu$ is currently being implemented. The hope is to mimic the continuation strategy of §3.4 to obtain accurate structural and thermodynamic properties of molecular fluids using the molecular interpolating closure for various densities.

**5. Conclusions.** Obtaining accurate structural and thermodynamic properties of fluids requires both integral equation theory and closure equations. Common closure equations from literature can result in inaccurate pair correlation functions and thermodynamic inconsistencies. Marucho and Pettitt [8] describe a new "interpolating" closure equation that improves upon existing closure equations for both atomic and molecular fluids, which adds an optimization constraint to the previous theories. We implement this new closure equation with a formulation that removes the optimization constraint in favor of augmenting the original system with additional equations. The result is a numerical procedure conducive to performing a continuation study in density for atomic fluids. Currently, we are working to do the same for molecular fluids. This will allow one to understand critical behavior of fluids that undergo density changes.

## REFERENCES

[1] M. Abramowitz and I. Stegun, *Handbook of Mathematical Functions*, Dover, New York, 1972.

[2] P. Attard, *Thermodynamics and Statistical Mechanics: Equilibrium by Entropy Maximisation*, Academic Press, Inc., London, 2002.

[3] K. Dickson, C. T. Kelley, B. Pettitt, A. Salinger, and J. Howard, *Distributions of atoms in fluids using LOCA*, Tech. Report SAND 2006-6564P, Sandia National Laboratories, October 2006.

[4] K. Dyer, J. Perkyns, and B. Pettitt, *Effective density terms in proper integral equations*, J. Chem. Phys., 123 (2005), p. 204512.

[5] J. P. Hansen and I. R. McDonald, *Theory of Simple Liquids*, Academic Press, Inc., London, second ed., 1986.

[6] H. Keller, *Lectures on Numerical Methods in Bifurcation Theory*, Tata Institute of Fundamental Research, Lectures on Mathematics and Physics, Springer-Verlag, New York, 1987.

[7] L. Luie and D. Blankschtein, J. Chem. Phys., 102 (1995), p. 5427.

[8] M.Marucho and B. M. Pettitt, *An optimized theory for simple and molecular fluids*, J. Chem. Phys., 126 (2007), pp. 124107–124107–9.

[9] A. Salinger and L. Frink, *Rapid analysis of phase behavior with density functional theory, Part I: Novel numerical methods*, J. Chem. Phys, 118 (2003), pp. 7457–7465.

# EVALUATION OF MAGNETIC VECTOR POTENTIAL IN 2D AND 3D MODELS

KENNY CHOWDHARY[*] AND ALLEN C. ROBINSON[†]

**Abstract.** In this paper, we discuss the derivation and implementation of a fast and accurate method for the evaluation of the magnetic vector potential associated with a circular loop current source field. We provide a quadratic interpolation polynomial to evaluate the vector potential for field points inside the loop. For field points outside the loop we use a rapidly converging method for elliptic integrals to evaluate the vector potential. We implemented this functionality into the ALEGRA code in order to provide 2D and 3D circular loop source fields.

**1. Introduction.** ALEGRA is a multi-material multi-physics shock hydrocode designed in an ALE framework. ALE (Arbitrary Lagrangian-Eulerian) is a numerical method that incorporates both Lagrangian meshes moving with the material as well as remeshing and remapping techniques to provide for meshes that must be smoothed or remain stationary. ALE begins with a Lagrangian method, which utilizes a moving mesh that follows the material. Next, when the elements become too distorted, a remeshing and remapping is performed. This sequence is repeated many times as the simulation proceeds. ALEGRA can be used to study problems in magnetohydrodynamics (MHD), which is the study of electrically conducting fluids, such as plasma. For example, ALEGRA can be used to study various aspect of experiments conducted on the Z-machine at Sandia National Laboratories such as magnetically driven flyer plates and wire-array implosions [3, 4]. In particular, ALEGRA can be used to study how electrically conducting fluids or solids move given an electromagnetic source term derived from a current source. In this paper, we will be concerned with magnetics problems for which we desire a source field generated by a circular loop current. For instance, the circular loop current is clearly applicable to a coil-gun geometry. We will accurately and efficiently derive the steady state magnetic vector potential source field that is induced by a circular loop current. This source field is then used as a forcing function and is one method to drive the magnetics diffusion portion of the ALEGRA computation. In particular, since the divergence of the magnetic flux density equals zero or $\nabla \cdot \mathbf{B} = 0$ then we must have a vector potential representation, $\mathbf{A}$, where $\mathbf{B} = \nabla \times \mathbf{A}$. The reduced form of Ampere's Law, $\nabla \times \mathbf{H} = \mathbf{J}$, Faraday's Law in the form $\mathbf{E} = -\frac{\partial \mathbf{A}}{\partial t}$, Ohm's law, $\mathbf{J} = \sigma \mathbf{E}$, where $\sigma$ is the conductivity, and $\mathbf{B} = \mu_0 \mathbf{H}$, where $\mu_0$ is the void permeability can be combined to obtain

$$\nabla \times \left(\frac{\nabla \times \mathbf{A}}{\mu_0}\right) = -\sigma \frac{\partial \mathbf{A}}{\partial t} + \mathbf{J}_s. \tag{1.1}$$

We define a source vector potential as a solution of

$$\nabla \times \left(\frac{\nabla \times \mathbf{A_s}}{\mu_0}\right) = \mathbf{J}_s. \tag{1.2}$$

The vector potential will respond to this given source potential and begin to diffuse throughout the conducting region at a rate determined by the conductivity.

**2. Circular Loop Current.** For a current loop of very small cross section in an infinite space, it can be shown that

$$\mathbf{A_s} = \frac{\mu_0 I}{4\pi} \oint \frac{d\mathbf{l}}{|\mathbf{r}|} \tag{2.1}$$

[*]Brown University,Division of Applied Mathematics, kchowdhary@brown.edu
[†]Sandia National Laboratories, acrobin@sandia.gov

and

$$\mathbf{B_s}(\mathbf{r}) = \frac{\mu_0 I}{4\pi}\int \frac{d\mathbf{l}\times\hat{\mathbf{r}}}{|\mathbf{r}|}, \tag{2.2}$$

where $\mathbf{B_s}$ is a vector field as a function of the field point $\mathbf{r}$, $\widehat{\mathbf{r}}$ is the unit vector from the evaluation point on the loop to the field point $\mathbf{r}$, and $d\mathbf{l}$ is the change in length along the loop (i.e. $d\mathbf{l} = dx\hat{\mathbf{i}} + dy\hat{\mathbf{j}} + dz\hat{\mathbf{k}}$)[2]. We will use this equation to analytically derive the source magnetic vector potential, $\mathbf{A_s}$, given a circular loop current.

## 3. Magnetic Vector Potential for a Circular Loop.

**3.1. Deriving the Integral.** We can rewrite equation (2.1)

$$\mathbf{A_s} = \frac{I}{4\pi}\oint \frac{d\mathbf{l}}{r_{pq}}$$

where $r_{pq} = |\mathbf{r}_p - \mathbf{r}_q|$, $\mathbf{r}_q$ is the position of the source current differential element, and $\mathbf{r}_p$ is a point in space.

Let us calculate the magnetic vector potential produced by a circular loop current with radius $a$. In cylindrical coordinates we have

$$\mathbf{r}_p = r\cos\theta\hat{\mathbf{i}} + r\sin\theta\hat{\mathbf{j}} + z\hat{\mathbf{k}}$$
$$\mathbf{r}_q = a\cos\varphi\hat{\mathbf{i}} + a\sin\varphi\hat{\mathbf{j}} + z_q\hat{\mathbf{k}}.$$

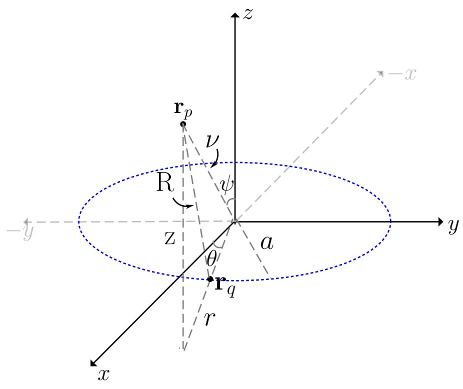


FIG. 3.1. *Illustration of the circular loop and the field point,* $\mathbf{r}_p$.

Then,

$$|\mathbf{r}_p - \mathbf{r}_q| = r^2 + a^2 - 2ra\cos(\theta-\varphi) + (z - z_q)^2$$

Let us assume $z_q = 0, \theta = 0$. Due to symmetry of the system, this assumption will allow us to calculate the vector potential at any point in space.

Next, for ease of notation, let $z^2 + r^2 = \nu^2$. Therefore, the above equation becomes

$$r_{pq}^2 = \nu^2 + a^2 - 2ra\cos\varphi.$$

Furthermore, we can write $r = \nu\sin\psi$, where $\psi$ is the angle between $\mathbf{r}_p$ and the $z$ axis. Thus,

$$r_{pq}^2 \quad = \quad \nu^2 + a^2 - 2a\nu\sin\psi\cos\varphi \ . \tag{3.1}$$

The path of integration is simply the circle, $\mathbf{l} = a\cos\varphi\hat{\mathbf{i}} + a\sin\varphi\hat{\mathbf{j}}$. Thus, $\mathbf{A_s}$ becomes

$$\mathbf{A_s} = \frac{\mu_0 I}{4\pi}\int_{-\pi}^{\pi}\frac{1}{r_{pq}}\left(-a\sin\varphi\hat{\mathbf{i}} + a\cos\varphi\hat{\mathbf{j}}\right)d\varphi. \tag{3.2}$$

Since $\sin\varphi$ is odd and $r_{pq}$ is even, the above equation simplifies to

$$\mathbf{A_s} = \frac{\mu_0 I}{2\pi}\int_0^{\pi}\frac{a\cos\varphi}{\sqrt{v^2 + a^2 - 2av\sin\psi\cos\varphi}}\hat{\mathbf{j}}d\varphi. \tag{3.3}$$

If we need the vector potential in a direction other than $\varphi = 0$, we can simply perform a change of basis.

**3.2. Evaluating the Integral.** Let us rewrite equation (3.3) into a more familiar form. First, let us perform a change of variables where $t = \varphi/2$. Then, $\frac{dt}{d\varphi} = \frac{1}{2}$ and we get

$$\frac{I}{2\pi}\int_0^{\pi}\frac{a\cos\varphi}{\sqrt{v^2 + a^2 - 2av\sin\psi\cos\varphi}}d\varphi = \frac{I}{\pi}\int_0^{\pi/2}\frac{a\cos 2t}{\sqrt{v^2 + a^2 - 2av\sin\psi\cos 2t}}dt.$$

Substituting in $\cos 2t = \cos^2 t - \sin^2 t$, the above equation becomes

$$\begin{aligned} RHS &= \frac{I}{\pi}\int_0^{\pi/2}\frac{a\cos 2t}{\sqrt{v^2 + a^2 - 2av\sin\psi(\cos^2 t - \sin^2 t)}}dt \\ &= \frac{I}{\pi}\int_0^{\pi/2}\frac{a\cos 2t}{\sqrt{(v^2 + a^2)(\cos^2 t + \sin^2 t) - 2av\sin\psi(\cos^2 t - \sin^2 t)}}dt \\ &= \frac{aI}{\pi}\int_0^{\pi/2}\frac{\cos 2t}{\sqrt{(v^2 + a^2 - 2av\sin\psi)\cos^2 t + (v^2 + a^2 + 2av\sin\psi)\sin^2 t}}dt. \end{aligned}$$

Now, let $b = \sqrt{v^2 + a^2 - 2av\sin\psi}, c = \sqrt{v^2 + a^2 + 2av\sin\psi}$ to get

$$I(b,c) = \frac{aI}{\pi}\int_0^{\pi/2}\frac{\cos 2t}{\sqrt{b^2\cos^2 t + c^2\sin^2 t}}dt. \tag{3.4}$$

This integral is called an elliptic integral, due to presence of $\sqrt{b^2\cos^2 t + c^2\sin^2 t}$, which is the radius of an ellipse in polar coordinates, centered at the origin and where $t$ is the angular coordinate. The study of evaluating such integrals is well known and is discussed in [5].[1]

To simplify equation (3.4) even further, we can write the numerator of the integrand as a function of the denominator. Let us call $R^2 = b^2\cos^2 t + c^2\sin^2 t$. We want to be able to write the numerator as some linear combination of $R^2$. That is, we want to find $\alpha_0$ and $\alpha_2$ such that $\alpha_0 + \alpha_2 R^2 = \cos^2 t$. We know that $\cos 2t = \cos^2 t - \sin^2 t$ and $1 = \cos^2 t + \sin^2 t$. So after after some algebra we get

$$\alpha_2 = \frac{2}{b^2 - c^2}, \alpha_0 = -\frac{b^2 + c^2}{b^2 - c^2}.$$

Therefore, (3.4) becomes

$$I(b,c) = \frac{aI}{\pi}\int_0^{\pi/2}\frac{H(R)}{R}dt \tag{3.5}$$

[1]Up and through section 3.2.3 is a detailed explanation of pages 254-260 in [5].

where $H(R) = \alpha_0 + \alpha_2 R^2$.

Now let us rewrite (3.4) in terms of $dR$. First, we have

$$\frac{dR}{dt} = \frac{(c^2 - b^2)(\sin t \cos t)}{R}.$$

Notice that this derivative is negative for $b < a$ since $\sin t$ and $\cos t$ are both positive on $[0, \frac{\pi}{2}]$ and so is $R$. Then, equation (3.5) turns into

$$\int_0^{\pi/2} \frac{H(R)}{R} dt = \int_0^{\pi/2} \frac{H(R)}{(c^2 - b^2)(\sin t \cos t)} \frac{dR}{dt} dt.$$

Furthermore, we can rewrite the denominator as $(R^2 - c^2)(b^2 - R^2)$. Thus,

$$\sqrt{(R^2 - c^2)(b^2 - R^2)} = |(b^2 - c^2)| \sin t \cos t$$

since $\sin t$ and $\cos t$ are positive on our domain. Furthermore, $|(b^2 - c^2)| \sin t \cos t = -(b^2 - c^2) \sin t \cos t = (c^2 - b^2) \sin t \cos t$ and if we let $\Delta(R) = \sqrt{(R^2 - c^2)(b^2 - R^2)}$ then

$$\int_0^{\pi/2} \frac{H(R)}{(c^2 - b^2)(\sin t \cos t)} \frac{dR}{dt} dt = \int_b^c \frac{H(R)}{\Delta(R)} dR \tag{3.6}$$

where $R(0) = b, R(\frac{\pi}{2}) = c$. If $b > c$ then $\Delta(R) = (b^2 - c^2) \sin t \cos t$,and we get

$$\int_0^{\pi/2} \frac{H(R)}{(c^2 - b^2)(\sin t \cos t)} \frac{dR}{dt} dt = \int_c^b \frac{H(R)}{\Delta(R)} dR. \tag{3.7}$$

In general, we have

$$\int_0^{\pi/2} \frac{H(R)}{R} dt = \int_{b \wedge c}^{b \vee c} \frac{H(R)}{\Delta(R)} dR,$$

where $\Delta(R)= \sqrt{(R^2 - c^2)(b^2 - R^2)} > 0$.

**3.2.1. Invariance of $\int_b^c \frac{H(R)}{\Delta(R)} dR$.** Let us show that equation (3.6) is invariant under the transformation $b \to \frac{b+c}{2}, c \to \sqrt{bc}$. Consider the change of variables given by $\hat{R} \doteq \frac{1}{2}(R + bc/R)$. Assume $c > b$ so that our integral is $\int_b^c \frac{H(R)}{\Delta(R)} dR$ (the case for $c < b$ is similar). Notice that for $R : [b, c]$, $\hat{R}$ is not injective. So we have to be careful when performing a change of variables.

We can rewrite the equation for $\hat{R}$ as $R^2 - 2\hat{R}R + bc = 0$. Then the equations for $R$ in terms of $\hat{R}$ become $R = \hat{R} + \sqrt{\widehat{R}^2 - bc}$ and $\hat{R} - \sqrt{\widehat{R}^2 - bc}$. By implicit differentiation, we get

$$\frac{dR}{d\hat{R}} = \frac{R}{R - \hat{R}} = \pm \frac{R}{\sqrt{\widehat{R}^2 - bc}}.$$

Define $b_1 = \frac{b+c}{2}, c_1 = \sqrt{bc}$. A little algebra shows that $\Delta^2(R) = 4R^2 \left(b_1^2 - \hat{R}^2\right)$ . If we define $\hat{\Delta}^2(\hat{R}) \doteq (\hat{R}^2 - c_1^2)(b_1^2 - \hat{R}^2)$ then

$$\frac{dR}{d\hat{R}} = \frac{\pm R}{\sqrt{\hat{R}^2 - bc}} = \frac{1}{2} \frac{\sqrt{4R^2(b_1^2 - \hat{R}^2)}}{\sqrt{(\hat{R}^2 - c_1^2)(b_1^2 - \hat{R}^2)}} = \pm \frac{\Delta}{2\hat{\Delta}} \quad .$$

Thus, $\frac{d\hat{R}}{dR} = \pm\frac{2\hat{\Delta}}{\Delta}$.

Rewriting our (3.6), we get

$$\int_b^c \frac{H(R)}{\Delta(R)}dR = \int_b^{\sqrt{bc}} \frac{H(R)}{\Delta(R)}dR + \int_{\sqrt{bc}}^b \frac{H(R)}{\Delta(R)}dR\,. \tag{3.8}$$

For $R : [b, \sqrt{bc}]$ we use the transformation $\hat{R} - \sqrt{\widehat{R}^2 - bc}$ and for $R : [\sqrt{bc}, c]$ we use the transformation $\hat{R} - \sqrt{\widehat{R}^2 - bc}$. Then,

$$\begin{aligned}\int_b^{\sqrt{bc}} \frac{H(R)}{\Delta(R)}dR &= -\int_{\frac{b+c}{2}}^{\sqrt{bc}} \frac{H(\hat{R} - \sqrt{\widehat{R}^2 - bc})}{\Delta(R)}\frac{\Delta}{2\hat{\Delta}}d\hat{R} \\ &= \int_{\sqrt{bc}}^{\frac{b+c}{2}} \frac{\frac{1}{2}H(\hat{R} - \sqrt{\hat{R}^2 - bc})}{\hat{\Delta}(\hat{R})}d\hat{R}.\end{aligned}$$

and

$$\int_{\sqrt{bc}}^c \frac{H(R)}{\Delta(R)}dR = \int_{\sqrt{bc}}^{\frac{b+c}{2}} \frac{\frac{1}{2}H(\hat{R} + \sqrt{\hat{R}^2 - bc})}{\hat{\Delta}(\hat{R})}d\hat{R}.$$

So now (3.6) becomes

$$\int_b^c \frac{H(R)}{\Delta(R)}dR = \int_{\sqrt{bc}}^{\frac{b+c}{2}} \frac{\frac{1}{2}\left(H(\hat{R} + \sqrt{\hat{R}^2 - bc}) + \frac{1}{2}H(\hat{R} - \sqrt{\hat{R}^2 - bc}\right)}{\hat{\Delta}(\hat{R})}d\hat{R} \tag{3.9}$$

If we define

$$H_1(R_1) \doteq \frac{1}{2}\left(H(R_1 + \sqrt{R_1^2 - bc}) + \frac{1}{2}H(R_1 - \sqrt{R_1^2 - bc})\right), b_1 = \frac{b+c}{2}, c_1 = \sqrt{bc}$$

then (3.9) becomes

$$\int_b^c \frac{H(R)}{\Delta(R)}dR = \int_{b_1}^{c_1} \frac{H_1(R_1)}{\Delta(R_1)}dR_1.$$

Remember that we showed $\int_0^{\pi/2} \frac{H(R)}{R}dt = \int_b^c \frac{H(R)}{\Delta(R)}dR$, for $H$ continuous. Hence,

$$\int_0^{\pi/2} \frac{H(R)}{R}dt \quad = \quad \int_b^c \frac{H(R)}{\Delta(R)}dR \quad = \quad \int_{b_1}^{c_1} \frac{H_1(R_1)}{\Delta(R_1)}dR_1 \quad = \quad \int_0^{\pi/2} \frac{H_1(R_1)}{R_1}dt.$$

Thus, the elliptic integral we started with is invariant under what is called the arithmetic-geometric transformation: $b \to b_1, c \to c_1$. Now, if we define

$$H_n(R_n) \doteq \frac{1}{2}\left(H_{n-1}(R_n + \sqrt{R_n^2 - bc}) + \frac{1}{2}H_{n-1}(R_n - \sqrt{R_n^2 - bc})\right),$$

where $b_n = \frac{b_{n-1}+c_{n-1}}{2}, c_n = \sqrt{b_{n-1}c_{n-1}}$ we get

$$\int_0^{\pi/2} \frac{H(R)}{R}dt = \lim_{n\to\infty}\int_0^{\pi/2} \frac{H_n(R_n)}{R_n}dt.$$

**3.2.2. Elliptic Integrals and the Arithmetic-Geometric Mean.** Let $H(R)$ be a polynomial over the reals. (i.e. $H(R) = \Sigma_{j=0}^{p} \alpha_j R^j$ where $\alpha_j \in \mathbb{R}$). Then,

$$\int_0^{\pi/2} \frac{H(R)}{R} dt = \int_0^{\pi/2} \lim_{n\to\infty} \frac{\sum_{j=0}^{p} \alpha_j(n) R_n^j(t)}{R_n(t)} dt \tag{3.10}$$

where

$$\alpha_j(n) = \begin{cases} \sum_{r=j}^{p} \alpha_r(n) c^{r-j}(n+1)\sigma(j,r), & 1 \leqslant j \leqslant p \\ \sum_{r=0}^{p} \alpha_r(n) c^{r}(n+1)\sigma(0,r), & j = 0 \end{cases}$$

$\alpha_j(0) = \alpha_j$ and $\sigma(j,r)$ is the coefficient of the $x^j$ term in $\frac{(x+\sqrt{x^2-1})^r + (x-\sqrt{x^2-1})^r}{2}$, assuming, of course, that the limit exists [5]. What happens to $R_n$ as $n \to \infty$? One nice property of the $b_n$'s and $c_n$'s is that $b < b_1 < \cdots < c_1 < c$. Thus, this sequence is monotonic and bounded. Hence, the limit exists, and in fact $\lim_{n\to\infty} b_n = \lim_{n\to\infty} c_n$. We call this limit the arithmetic-geometric mean and we get

$$\lim_{n\to\infty} R_n = \lim_{n\to\infty} \sqrt{b_n^2 \cos^2 t + c_n^2 \sin^2 t} = \lim_{n\to\infty} \sqrt{b_n^2 \cos^2 t + c_n^2 \sin^2 t} = \lim_{n\to\infty} b_n.$$

So the limit of the integrand can be written strictly in terms of $b_n$:

$$\lim_{n\to\infty} \frac{\sum_{j=0}^{p} \alpha_j(n) R_n^j(t)}{R_n(t)} = \lim_{n\to\infty} \frac{\sum_{j=0}^{p} \alpha_j(n) b_n^j}{b_n}.$$

Now we can see the integrand is independent of $t$ so that

$$\int_0^{\pi/2} \frac{H(R)}{R} dt = \lim_{n\to\infty} \frac{\sum_{j=0}^{p} \alpha_j(n) b_n^j}{b_n} \frac{\pi}{2}. \tag{3.11}$$

**3.2.3. Convergence of the limit.** Let us show that the integrand in (3.10) converges as $n \to \infty$. Recall that we can write the vector potential as

$$I(b,c) = \frac{aI}{\pi} \int_0^{\pi/2} \frac{H(R)}{R} dt, \tag{3.12}$$

where $H(R) = \alpha_0 + \alpha_2 R^2$ and $\alpha_2 = \frac{2}{b^2-c^2}, \alpha_0 = -\frac{b^2+c^2}{b^2-c^2}$.

The first iteration gives us

$$H_1(R_1) = (\alpha_0 - \alpha_2 bc) + 2\alpha_2 R_1^2.$$

Thus, $\alpha_0(n+1) = \alpha_0(n) - \alpha_2(n) b_{n+1}^2$ and $\alpha_2(n+1) = 2\alpha_2(n)$ where $\alpha_0(0) = a_0$ and $\alpha_2(0) = a_2$. So we get $\alpha_2(n+1) = 2^{n+1}\alpha_2(0)$. Then, $\alpha_0(n+1)$ becomes $\alpha_0(n) - 2^n \alpha_2(0) b_{n+1}^2$. We know that $c_{n+1}^2 \to b_\infty$, a finite constant independent of $t$. Therefore, $\alpha_0(n+1) \sim \alpha_0(n) - 2^n \alpha_2(0) b_\infty^2$ and the general solution to this difference equation becomes $\alpha_0(n) = \beta - 2^n \alpha_2(0) c_\infty^2$, where $\beta$ is a constant (it is the solution to the homogeneous equation $\alpha_0(n+1) = \alpha_0(n)$). Now it is clear that $\alpha_0(n) + \alpha_2(n) R_n^2$ converges in the limit. Hence, the solution to (3.5) is

$$I(b,c) = \frac{aI}{\pi} \lim_{n\to\infty} \frac{\pi}{2} \frac{\alpha_0(n) + \alpha_2(n) b_n^j}{b_n}. \tag{3.13}$$

**3.2.4. Coding up the Integral.** Using the recursive algorithm in (3.13) with $\alpha_0(n+1) = \alpha_0(n) - \alpha_2(n)b_{n+1}^2$ and $\alpha_2(n+1) = 2\alpha_2(n)$ we can solve for the limit once we determine $\alpha_0(0)$ and $\alpha_2(0)$. Since $\alpha_0(0) = -\frac{b^2+c^2}{b^2-c^2}$ and $\alpha_2(0) = \frac{2}{b^2-c^2}$ we get

$$\alpha_2 = \frac{-1}{2av\sin\psi}$$
$$\alpha_0(0) = -\alpha_2 v^2 - a^2.$$

Because of quadratic convergence of this algorithm, we do not need to take large values of $n$ to approximate the integral [5].

**3.3. Magnetic Vector Potential within the loop: 2D model.** So far, we have assumed that the current has been running along an infinitesimally small volume in a circular path. This of course leads to a singularity when the field point approaches any of the source points (see equation (2.1)). What is really happening, however, is that the current is flowing through a wire with non-zero volume. Let us assume that the current loop has radius $\epsilon$ and the current density in the loop is $I/(\pi a^2)$. How do we calculate the vector potential within the circular loop source? This is a common problem for numerical methods based on free space Green's functions and some sort of regularization is always required to handle the singularity. Furthermore, since we are using the potential as a driving field we know that any assumption that we make about the current density in the wire is really not correct. We must make some simple, fast and useful approximation to compute the potential whenever the field point is close to a source point. Outside the loop, the analytic expression derived above gives an excellent value of the vector potential. We assume that as we approach the loop the vector potential is given by this outer solution. However, when we are inside the loop, we will find a relation between the vector potential and **J**.

Recall that $\nabla\times(\nabla\times\mathbf{A}) = \nabla\times\mathbf{B} = \mathbf{J}$, where **J** is our current density and **B** is the magnetic field. For $\mathbf{J} \neq \mathbf{0}$ within the loop, this gives us a differential equation for the vector potential, **A**. Let us calculate this differential equation. Recall that **A** has a component only in the $\hat{\varphi}$ direction and is a function of $r$ and $z$ ($v = \sqrt{r^2+z^2}$). Then,

$$\nabla\times\mathbf{A} = -\frac{\partial A_\varphi}{\partial z}\hat{\mathbf{r}} + \left(\frac{\partial A_\varphi}{\partial r} + \frac{A_\varphi}{r}\right)\mathbf{z}$$

Therefore,

$$\nabla\times(\nabla\times\mathbf{A}) = \left(-\frac{\partial^2 A_\varphi}{\partial z^2} - \frac{\partial^2 A_\varphi}{\partial r^2} - \frac{1}{r}\frac{\partial A_\varphi}{\partial r} + \frac{A_\varphi}{r^2}\right)\hat{\varphi}$$

Hence,

$$-\mathbf{J} = \left(\frac{\partial^2 A_\varphi}{\partial z^2} + \frac{\partial^2 A_\varphi}{\partial r^2} + \frac{1}{r}\frac{\partial A_\varphi}{\partial r} - \frac{A_\varphi}{r^2}\right)\hat{\varphi}. \tag{3.14}$$

For points within the loop, we will use a quadratic interpolating polynomial in $\mathbb{R}^2$ to approximate the vector potential. The key here will be to use the above differential equation to determine the coefficients of this interpolating polynomial.

Let us take a cross section of the wire in the $yz$ plane (see Figure 3.2). Our goal is to obtain a constant coefficient interpolating polynomial of the form

$$A_\varphi = A_{0,0} + \frac{\partial A_\varphi}{\partial r}(r-a) + \frac{\partial A_\varphi}{\partial z}(z-c_0) + \frac{1}{2}\frac{\partial^2 A_\varphi}{\partial r^2}(r-a)^2 + \frac{1}{2}\frac{\partial^2 A_\varphi}{\partial z^2}(z-c_0)^2 \tag{3.15}$$

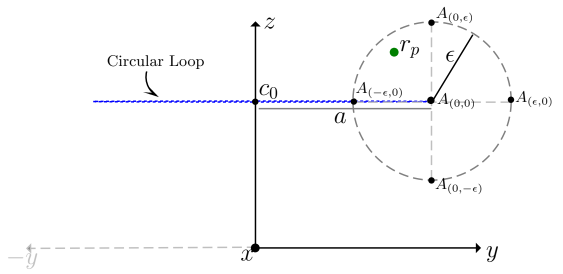


FIG. 3.2. *This is a cross-section in the yz plane of a circular loop centered about the z axis.*

The value of the potential at the center of this cross-section is $A_{0,0}$, where the subscripts represent local $r$ and $z$ coordinates, respectively. We can approximate these coefficients by first computing values for $A_{\varepsilon,0}$, $A_{-\varepsilon,0}$, $A_{0,\varepsilon}$, and $A_{0,-\varepsilon}$ from equation 3.3. It follows that

$$\frac{\partial^2 A_\varphi}{\partial z^2} = \frac{A_{0,\varepsilon} - 2A_{0,0} + A_{0,-\varepsilon}}{\varepsilon^2},$$
$$\frac{\partial^2 A_\varphi}{\partial r^2} = \frac{A_{\varepsilon,0} - 2A_{0,0} + A_{-\varepsilon,0}}{\varepsilon^2},$$
$$\frac{\partial A_\varphi}{\partial r} = \frac{A_{\varepsilon,0} - A_{-\varepsilon,0}}{2\varepsilon},$$
$$\frac{\partial A_\varphi}{\partial z} = \frac{A_{0,\varepsilon} - A_{0,-\varepsilon}}{2\varepsilon}.$$

It remains to show how to determine $A_{0,0}$. Using (3.14) and plugging in our quadratic interpolant at $(a, c_0)$, we obtain

$$\frac{A_{0,\varepsilon} - 2A_{0,0} + A_{0,-\varepsilon}}{\varepsilon^2} + \frac{A_{\varepsilon,0} - 2A_{0,0} + A_{-\varepsilon,0}}{\varepsilon^2} + \frac{1}{a}\frac{A_{\varepsilon,0} - A_{-\varepsilon,0}}{2\varepsilon} - \frac{A_{0,0}}{a^2} = -J.$$

where $J = \frac{I}{\pi\varepsilon^2}$. Solving for $A_{0,0}$ we obtain

$$A_{0,0} = \left(\frac{I}{\pi} + (A_{0,\epsilon} + A_{0,-\epsilon}) + (A_{\epsilon,0} + A_{-\epsilon,0}) + \frac{\epsilon}{2a}(A_{\epsilon,0} - A_{-\epsilon,0})\right)\left(4 + \frac{\epsilon^2}{a^2}\right)^{-1}$$

Figure 3.3 illustrates the vector potential for the cross-section shown in Figure 3.2.

In general, we can evaluate the vector potential for any field point in a 3D model (Figure 3.4(a)), by transforming the problem into a 2D model (Figure 3.4(b))by taking the appropriate slice along the radial direction of the loop.

## 4. Implementation.

### 4.1. Change of Basis.

Up till now, we have been using a circular loop source that is centered about the $z$ axis, parallel to the $xy$ plane (see Figure 4.1(a)). What if we have a circular loop that is angled or tilted as in Figure 4.1(b). If we choose an arbitrary field point in 3D space, how are we going to calculate the vector potential? First, we have to consider whether the field point is within the radius of the loop or not. If this distance is less than the

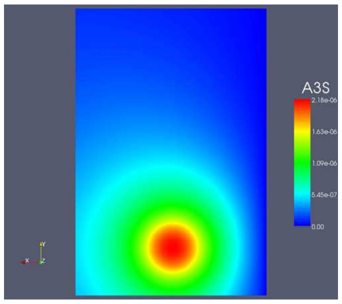
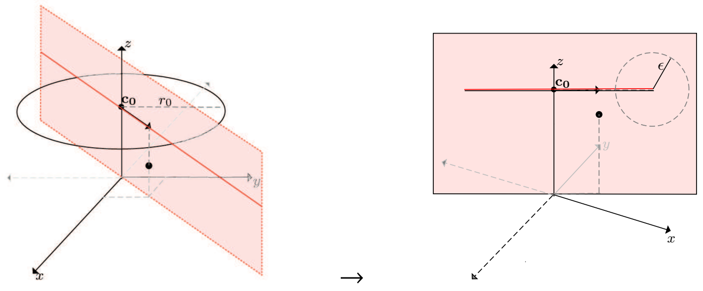


FIG. 3.3. *Illustration of the magnitude of the magnetic vector potential for a 2D model.*



(a) 3D problem (b) 2D problem

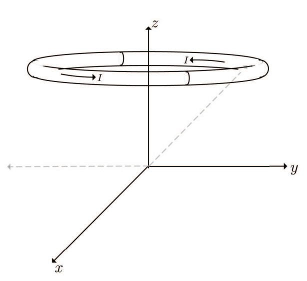
FIG. 3.4. *Transforming a 3D problem into a 2D problem*

radius, the field point is within the loop and we can use a quadratic interpolant to determine the vector potential. If the distance is greater than the radius, the field point is outside the loop and we can use our recursive algorithm to evaluate the vector potential.

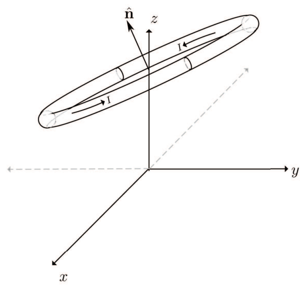
For a field point within the loop, in order to obtain a quadratic interpolant we need to determine the point on the loop closest to the field point (i.e. we need to determine the location of $A_{0,0}$) and the location of the points $A_{\varepsilon,0}, A_{-\varepsilon,0}, A_{0,\varepsilon}$, and $A_{0,-\varepsilon}$. For a field point outside the loop, we need to determine the relative distance from the center of the loop to the field point (i.e. we need to calculate $v$ and $\sin\psi$). In both cases, we can perform a change of basis to transform 4.1(b) into 4.1(a) in order to determine these critical points.

Given the normal vector to the loop's plane, $\hat{\mathbf{n}}$, we can find a perpendicular vector, $\mathbf{x}'$, satisfying $\hat{\mathbf{n}} \cdot (\mathbf{x}' - \mathbf{c_0}) = 0$, where $\mathbf{c_0}$ is the center of the loop. Once we have $\mathbf{x}'$, we can find another vector, $\mathbf{y}'$, orthogonal to $\mathbf{x}'$ and $\hat{\mathbf{n}}$, given by $\hat{\mathbf{n}} \times \mathbf{x}'$. Normalizing $\mathbf{x}'$ and $\mathbf{y}'$, and using $\hat{\mathbf{n}}$ for our new $\mathbf{z}$ axis, we have our desired orthonormal basis for the angled loop. Now, for example, we can easily compute any point on the angled loop with the equation $(x')^2 + (y')^2 = a^2$.

**4.2. Results.** The figures in 4.2 illustrate the magnitude of the vector potential for three different circular loop orientations. Figure 4.2(a) illustrates the magnitude of the vector potential given a circular loop that lies parallel to the $xy$ plane and is centered at the point $(0, 0, .5)$. Figure 4.2(b) shows the magnitude of the vector potential for a loop parallel to the $xy$ plane, but centered at $(.5, 0, .5)$. Finally, Figure 4.2(c) shows the magnitude of the vector potential for a circular loop centered at $(.5, 0, .5)$, whose normal to the loop's plane is the vector $(1, 0, 1)$. Clearly, we can see a high magnitude band, shown in red, for field points within the loop, which is appropriate given that the integrand in equation (3.3) is greatest for exactly



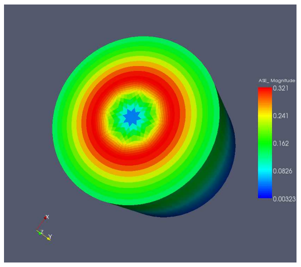
(a) Flat circular loop



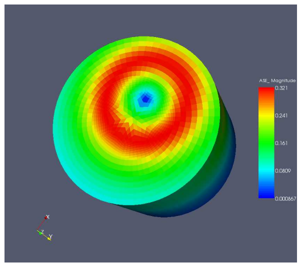
(b) Angled circular loop

FIG. 4.1. *Different orientations of the circular loop.*

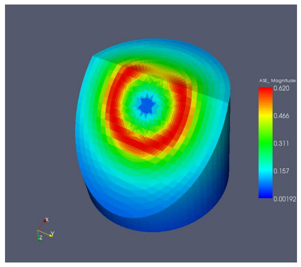
these points.



(a) Flat Loop



(b) Off-centered Loop



(c) Angled Loop

FIG. 4.2. *Magnitude of the vector potential for different circular loop orientations*

**4.3. Error and Performance.** Prior to implementing the methods outlined in this paper, we used the midpoint integration method to approximate equation (3.3) for field points outside the loop, introducing a regularization to handle the singularity as the field points approach the core of the loop. While the convergence for the midpoint method is of order $h^2$, our new method provides a much simpler, quadratically converging, recursive algorithm to obtain an exact solution to the vector potential, with an error on the order of $10^{-10}$ [1].

For field points approaching the interior of the loop, a smaller $h$ value was needed to obtain an approximation that differed from the true vector potential (i.e. the value obtained using the recursive algorithm) by an order of $10^{-10}$, whereas field points further away from the interior of the loop produced approximations on the same order of accuracy using significantly larger $h$ values. Thus, in order to obtain an error on the order of $10^{-10}$ using the midpoint method, we need to choose a relatively small $h$ value for all evaluation points. The recursive algorithm, however produces an exact value to the vector potential at all points outside the loop, with an error of $10^{-10}$, in at most ten iterations! The simplicity and quickness of the recursive algorithm resulted in a significantly faster computation time for the evaluation of the vector potential.

In the table below, we compare the computation times for the evaluation of the vector potential between the midpoint method and the recursive algorithm. The $h$ used in the midpoint method is chosen small enough so as to differ from the exact solution (the solution obtained using our recursive algorithm) by at worst $10^{-10}$ at all evaluation points.

The first column describes the number of of field points to be evaluated and the last

TABLE 4.1
*Performance times for the $A_\varphi$ Calculation*

| | User CPU Time (sec) | | |
|---|---|---|---|
| Field Points | Recursive Algorithm | Midpoint Method | Ratio |
| $2.842 \times 10^4$ | $7.000 \times 10^{-2}$ | $5.400 \times 10^{-1}$ | 7.71 |
| $6.286 \times 10^4$ | $1.100 \times 10^{-1}$ | 1.200 | 10.91 |
| $2.170 \times 10^5$ | $3.400 \times 10^{-1}$ | 4.300 | 12.65 |
| $7.789 \times 10^5$ | 1.430 | $1.491 \times 10^1$ | 10.43 |
| $1.604 \times 10^6$ | 3.300 | $3.032 \times 10^1$ | 9.19 |
| (Avg) | | | 10.18 |

column shows the ratio between the evaluation times of the midpoint method and the recursive algorithm. On average, the recursive algorithm is about 10 times faster!

Note that the regularized kernel approach does not attempt to carefully approximate the current density within the loop. Thus, not only does our method provide a faster and more accurate computation for field points outside the loop, but our quadratic interpolant provides an improved approximation for field points within the loop.

**5. Conclusions.** In this paper, we discussed the derivation and implementation of a fast and accurate method for the evaluation of a vector potential associated with a circular loop current source field. We provided a quadratic interpolation polynomial to evaluate the vector potential for field points inside the finite cored circular loop. For field points outside the loop, we used a rapidly converging method associated with elliptic integrals to evaluate the vector potential. We have implemented this functionality into the ALEGRA code in order to provide 2D and 3D circular loop source fields for magnetic diffusion simulations.

## REFERENCES

[1] G. ALMKVIST AND B. BERNDT, *Gauss, Landen, Ramanujan, the arithmetic-geometric mean, ellipses, π, and the Ladies Diary*, American Mathematical Monthly, August-September (1988), pp. 585–608.

[2] D. J. GRIFFITHS, *Introduction to Electrodynamics: Third Edition*, Prentice Hall, 1999.

[3] T. MEHLHORN, T. BRUNNER, M. DESJARLAIS, C. GARASI, T. HAILL, H. HANSHAW, R. LEMKE, T. MATTSSON, M. MATZEN, B. OLIVER, A. ROBINSON, S. SLUTZ, T.G.TRUCANO, E. YU, R. VESEY, M. CUNEO, B. JONES, M. D. KNUDSON, AND D. SINARS, *Towards a predictive MHD simulation capability for designing hypervelocity magnetically-driven flyer plates and PW class z-pinch x-ray sources on Z and ZR.* Proceedings Megagauss XI Conference, Imperial College, London, England, 2006.

[4] A. C. ROBINSON AND C. J. GARASI, *Three-dimensional z-pinch wire array modeling with ALEGRA-HEDP*, Computer Physics Communications, 164 (2004), pp. 408–413.

[5] J. WIMP, *Computation with Recurrence Relations*, Pitman Advanced Publishing Program, 1984.